Implementing a UART Function With the 8-Bit Interval Timer/Counter

Mark E. Buccini — *Mixed Signal Products*

ABSTRACT

This application report describes the implementation of a hardware/software universal asynchronous receiver transmitter (UART) function on the MSP430x31x, MSP430x32x, and MSP430x33x families of 16-bit RISC-like mixed-signal processors using the integrated 8-bit interval timer/counter (8-bit T/C). The UART function described in this report is half-duplex, event-driven, and supports an 8N1 protocol using an RS232 interface. Baud rates from 1,200–115,200 baud are possible.

Contents

List of Figures

1 Introduction

Many applications require serial communications between two systems. The implementation of a hardware/software UART function using the 8-bit T/C peripheral integrated into the MSP430x3xx families allows easy serial-communication operation (see Figure 1). While it is not always obvious that the flexible 8-bit T/C peripheral on the MSP430x3xx can be configured as a UART, this peripheral, when combined with software, provides all the major requirements to implement a complete, ultra-low power, and cost effective UART solution. The 8-bit T/C UART function described in this report offers the following key features:

- Automatic start-bit detect even from ultra-low power modes
- Hardware baud-rate generation and latching of RXD and TXD data
- Baud rates of 1,200 to 115,200 baud are possible

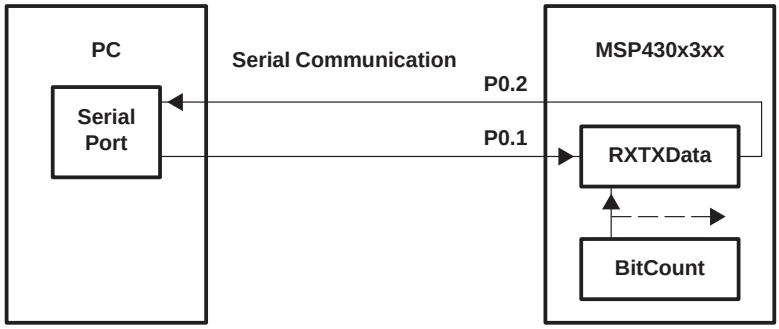

Figure 1. MSP430x3xx UART Function

2 Theory of Operation

IMPORTANT:
Review the current MSP430x31x, MSP430x32x, and MSP430x33x-family data sheets, and the *MSP430 Family Architecture and Module Guide* for exact MSP430 device specifications and module descriptions.

The MSP430x3xx communicates serially with a PC in this report. Characters are exchanged between the two systems over a receive/transmit line. The character protocol used is the common 8N1: 8 data bits, no parity, and one stop bit. Data is transferred between the MSP430 and the PC at 9,600-baud over an RS232 connection. Other protocols and baud rates can be implemented, including parity and 9^{th}-bit addressing. The UART function specifically described uses the 8-bit T/C and port pins P0.1 and P0.2. Inspection of the 8-bit T/C reveals that on all MSP430x3xx family members, P0.1 is multiplexed into a receive-data flip-flop (RXD_FF), and P0.2 is multiplexed into a transmit-data flip-flop (TXD_FF). Data latched in RXD_FF and TXD_FF are readable in the 8-bit T/C control register (TCCTL) as bits RXD and TXD, respectively. When enabled, the overflow of the 8-bit T/C automatically latches RXD in and TXD out irrespective of other hardware and software execution. Therefore, software and interrupt latency associated with the UART function is not of great concern because the 8-bit T/C hardware does the actual transfer of RXD and TXD bits using the exact timing. UART-application software is responsible for preparing the RXD after latching and the TXD bits before latching. In this report, CPU register R5 is used for RXTXData, a buffer that holds the data being received or transmitted. Register R6 is used for BitCnt, a bit-tracking register. The selection of R5 and R6 is arbitrary—any CPU registers or RAM bytes can be used.

In receive mode, the 8-bit T/C is configured such that the falling edge of P0.1 automatically enables the 8-bit timer independent of real time software—the falling edge of P0.1 indicates a start bit. A single interrupt vector is used for the 8-bit T/C overflow and P0.1—in this example, this interrupt is enabled by the 8-bit T/C overflow. The MSP430x3xx can be idle or performing other tasks while the UART function is ready to receive data. No CPU resources are exercised until after the 8-bit timer has automatically been enabled and the first overflow occurs latching the first data bit, specifically from P0.1 into RXD—an interrupt is then issued to the CPU. The 8-bit T/C will continue to count, reloading a prescale value from the timer/counter preload register (TCPLD). The next 8-bit T/C overflow occurs in the middle of the next data bit. Software receives each RXD bit into the RXTXData buffer after the bit has been latched.

In transmit mode, each data bit from the RXTXData buffer is moved by software to TXD, which transmits specifically on pin P0.2. The overflow of the 8-bit T/C automatically latches out the TXD data on P0.2. The 8-bit T/C is reloaded automatically from TCPLD and continues to count. With the 8-bit T/C hardware automatically outputting TXD bits, interrupt latency and bit-timing concerns associated with pure software UARTs are largely eliminated.

2.1 MSP430x3xx Resources Used

MSP430x3xx provides several features to support the UART function:

- Port pin P0.1 used to receive data, and pin P0.2 used to transmit data
- An 8-bit T/C peripheral
- RXTXData register to buffer RXD and TXD serial data
- BitCnt register to track the progress of bits being received or transmitted

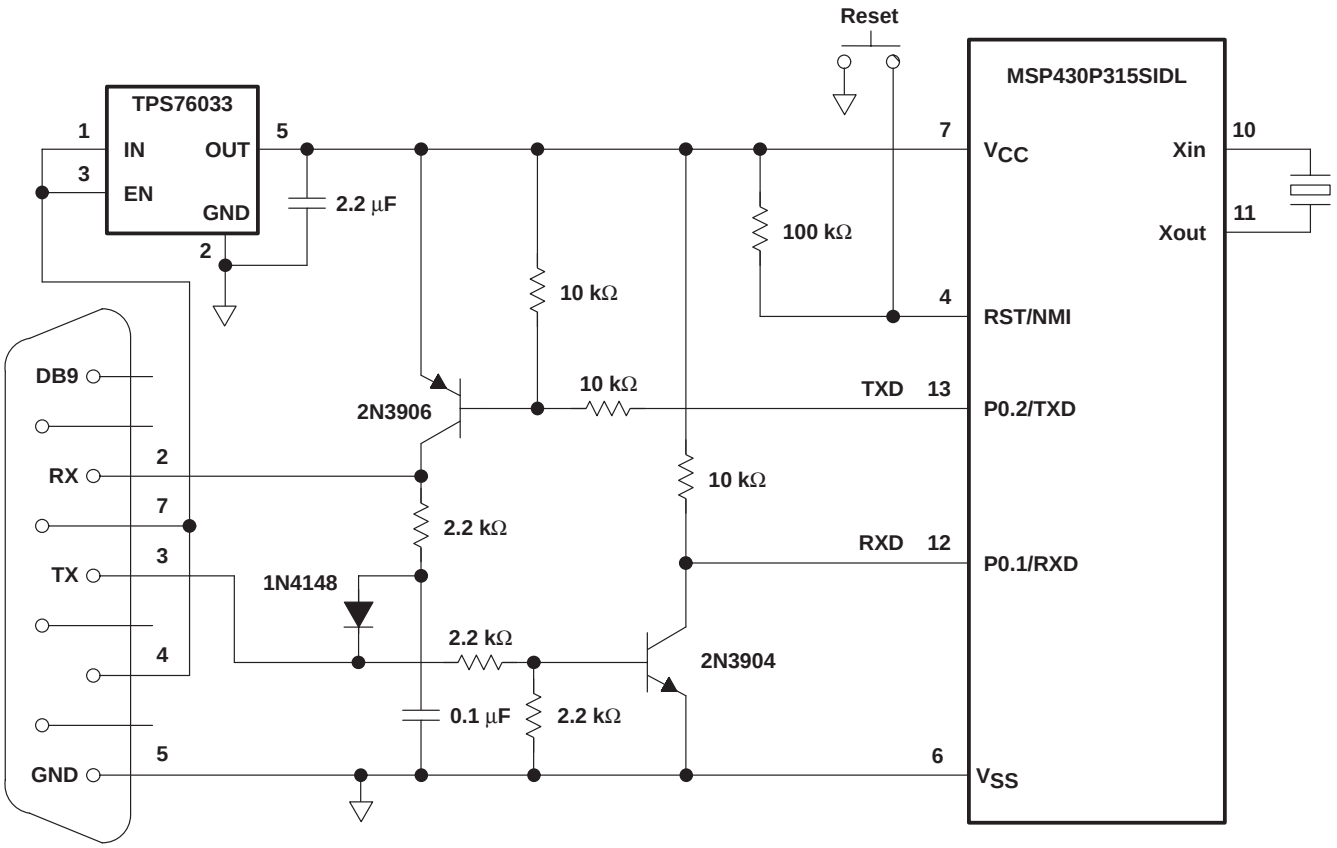

Figure 2. MSP430x31x UART Function Test Circuit

The 8-bit T/C peripheral is configured using TCCTL. Additionally, TCPLD is loaded with the exact timer interval to generate the required baud rate. TCPLD automatically loads and reloads the actual 8-bit timer/count data (TCDAT). To configure P0.1 to falling-edge enable (a typical start-bit condition), bit P0IES_1 is set in the P0 interrupt-edge-select register (P0IES), and P0.1 interrupt enable (P0IE1) is set in interrupt-enable register 1 (IE1).

Two dedicated software registers are required to support the UART function, RXTXData and BitCnt. RXTXData is basically a shift register used to buffer the data being received or transmitted—any available CPU register or RAM byte/word can be used. In this report, the same register is used for both receive and transmit. BitCnt is a register used to determine what bit is being received or transmitted. The software provided in listings *tc_uart1.asm* and *tc_uart2.asm* uses BitCnt as an autoincrement indirect pointer to an exact transmit or receive interrupt-service routine. With this method, the software does not have to determine what bit is being transmitted or received, saving CPU cycles and reducing time spent in the interrupt-service routine. BitCnt must be a CPU register during execution, as required by the auto-increment indirect addressing used. The software in listing *tc_uart4.asm* uses BitCnt as a simple counter to determine the status of the bit transmission. Autoincrement addressing is not used in this example, so BitCnt is just a RAM location. In the same *tc_uart4.asm* listing, RXTXData is a RAM location, not a CPU register as in the other examples.

2.2 External Hardware Description

All the external hardware required is for line-level translation associated with the RS232 interface to the PC. With RS232, a logic 1 is transmitted as less than –3 V, and a logic 0 as greater than 3 V. To perform RS232 line-level translation, integrated circuits such as TI's MAX232 provide a compliant solution. A simple low-cost, low-power solution is offered in Figure 2 using the MSP430P315SIDL configuration. For normal operation, the reset pin of the MSP430 must be pulled high. In this example, reset is pulled high through a 100 kΩ resistor, and a switch to ground is added for a hard reset if required. V_{CC} can be supplied from a battery. Due to the ultra-low power of the MSP430, the system is powered directly from the RS232 interface and regulated using a TPS76033. A 32,768-Hz watch crystal is used for the auxiliary clock (ACLK) generation. The master clock (MCLK) is user-programmable and generated from the MSP430x3xx digitally-controlled oscillator (DCO).

2.3 Baud Rate Calculation

The 8-bit T/C is configured for baud-rate generation. Based on the baud rate required, an interval bitime is calculated. Bitime is the length, in 8-bits, that T/C counts between individual bits. Bitime is the interval during which an 8-bit T/C will latch RXD and TXD data bits, and is simply the 8-bit T/C-clock source divided by the baud rate (see Figure 3). The calculated bitime is loaded into TCPLD, which automatically loads the 8-bit T/C counter. The MSP430x3xx MCLK is used on high-speed UARTs.

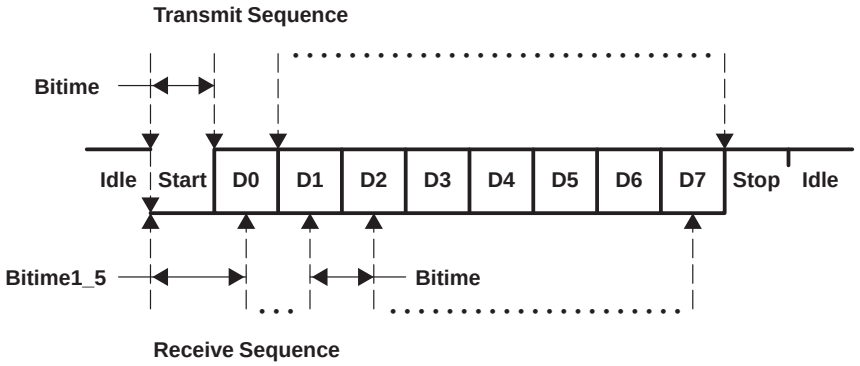

Figure 3. UART 8N1 Character

Example 1: 9,600-baud, MCLK at 1,048,576 Hz, and the 8-bit T/C clock source

Bitime= 109.2267 = 1,048,576 / 9,600 = clock source / baud rate

Use 109 for bitime

Actual baud rate = 1,048,576 / 109= 9,619 baud

Only an integer value of bitime can be moved to TCPLD for bit timing. The value 109 is used for bitime, assuming 9,600-baud and a 1,048,576-Hz MCLK clock source. The error of rounding bitime to the nearest integer is insignificant if the clock source is sufficiently large. In this example, the *actual baud rate* has an error of less than 0.2% per bit. If a different clock source or baud rate is required, bitime can be recalculated using the previous formula.

Let us look at another example of bitime calculation:

Example 2: 115,200-baud, MCLK at 3,244,032 Hz, and the 8-bit T/C clock source:

Bitime = 3,244,032 / 115,200 = 28.16 = clock source / baud rate

Use 28 for bitime

Actual baud rate = 3,244,032 / 28 = 115,858 baud

The firmware engineer can adjust MCLK from the default of 1,048,576 HZ as multiples of ACLK. In Example 2, MCLK is adjusted to 3,244,032 Hz (99 x 32,768). The system-clock frequency-control register (SCFQCTL) is loaded with the appropriate ACLK clock multiplier. It is possible that the nominal frequency bits (FN_2, FN_3, and FN_4) in the system-clock frequency-integrator register 0 (SCFI0) may also require modification to insure that the DCO is operating at a center tap. FN_2 is used when MCLK operates at approximately 2 MHz, FN_3 is used for MCLK operation at approximately 3 MHz, and FN_4 is used for MCLK operation in the 4-MHz range.

3 Software Description Listing tc_uart1.asm

Software Listing *tc_uart1.asm* is divided in several subroutines.

- Init_Sys initializes the MSP430
- TX_byte, when called, starts the UART to transmit one byte from RXTXData buffer. As coded, TX_byte will not return from the call until a complete byte is transmitted from RXTXData. This prevents the accidental overruns of starting a transmission while the UART is still actively transmitting previous data. Other polling techniques can be used.
- RX_ready, when called, gets the UART ready to receive one byte into the RXTXData buffer. Once readied, the CPU and the system are free for other tasks. The UART automatically receives data sent to RXTXData.

On RESET, an Init_Sys subroutine is called to initialize the system. Mainloop sends a **>** prompt to the PC and then waits to receive a character from the PC, which is echoed back (see Figure 4). The prompt is first sent to the PC by placing **>** in RXTXData and calling the TX_Byte subroutine. Next, the UART is placed in receive mode by calling the RX_Byte subroutine. The MSP430 waits in LPM0 by turning off the CPU. mainloop program execution holds at this point, but the UART is ready. On reception of a complete character into RXTXData, the UART software, as implemented, reactivates the CPU in the mainloop. The mainloop then calls TX_Byte to echo back the exact same character still in RXTXData. To transmit, the software simply moves the character to be transferred to RXTXData and calls the TX_Byte subroutine. To receive a data character, the RX_Byte subroutine is simply called to ready the UART function. Complete software is provided in software listing *tc_uart1.asm*.

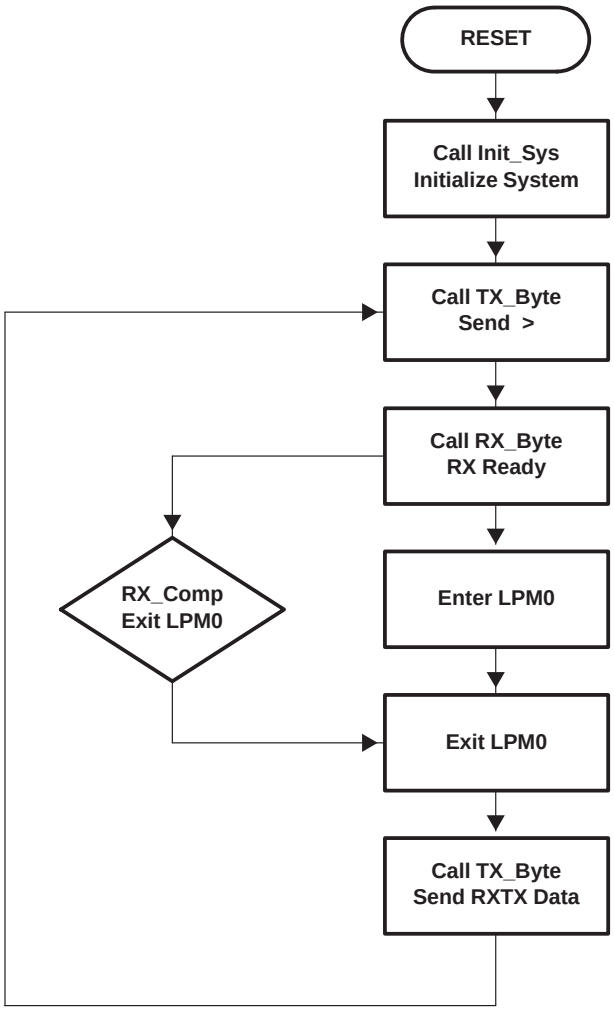

Figure 4. UART Function Software in Listing tc_uart1.asm

3.1 Software Overhead for the UART Function

A firmware engineer might be concerned by the system overhead of using a hardware/software UART. The function described in this report requires some CPU resources. The most important point is that no CPU resources are required when the UART function is ready to receive—even though the hardware/software UART is completely ready. Superfluous scanning for a start-bit condition is not required—the 8-bit T/C will automatically be enabled in the receive mode on a start-bit condition. In both received and transmit modes, the 8-bit T/C hardware also latches data bits being transferred.

Example: CPU overhead at 9,600-baud using a 1,048,576-Hz 8-bit T/C clock source during the actual data transfers:

15.6% receive mode—17 CPU-cycles are required to receive a bit.

18.3% transmit mode—20 CPU-cycles are required to transmit a bit.

The software in listing *tc_uart1.asm* uses autoincrement indirect addressing in the UART ISR:

```
mov     @BitCnt+,PC         ; Branch To Routine
```

Autoincrement indirect addressing allows immediate determination of the status of the UART ISR (that is, RX_Edge, Rx_Bit, and TX_Bit). The software does not have to determine its location in the receive or transmit sequence. CPU cycles spent in the UART ISR are greatly reduced, allowing time for other tasks. Based on the previous example, the firmware engineer must ensure that, if the UART function is to be available, the burden on the CPU does not exceed 82% at any point during real-time code execution. When the UART function (9,600-baud) is used with a 1,048,576 MCLK, approximately 859,532 16-bit operations per second remain available for other real-time activities.

3.2 Use of TI Standard Peripheral Definitions

All examples in this report use TI MSP430 standard peripheral definitions. The file *std_def.asm* is included with the ADT430 simulation environment, or it is available from TI's web site (www.ti.com). The file *std_def.asm* should be in the same working directory as the source-code examples included when compiled. The following assembler directive copies *std_def.asm* into the source:

```
.include     "STD_DEF.ASM"      ; Standard Definitions Used
```

TI encourages programmers to use TI standard peripheral definitions to promote commonality with the *MSP430 Architecture Guide and Module Library* and to simplify code debugging.

4 Using the UART Function in Ultra-Low Power Applications

It is possible to use the MSP430x3xx 8-bit T/C UART function during MSP430 advanced ultra-low power modes. It is also possible to generate high-baud rates when using only a low-frequency 32,768-Hz watch crystal. The MSP430's unique on-chip high-speed DCO generates the high speed MCLK, which is digitally stabilized automatically in MSP430x3xx family to a software-selectable multiple of the 32,768-Hz watch crystal using the on-chip frequency locked loop (FLL). The default MCLK on the MSP430x3xx has a frequency of 1,048,576 Hz, which is the frequency used in this report. The MCLK can be used for baud generation when selected as the clock source for the 8-bit T/C. The DCO starts and becomes stable in less than 6 μs. With a fast DCO start, baud rate generation is possible in ultra-low power modes, even with the DCO initially off—the falling edge of a start bit can automatically start the DCO and UART functions.

4.1 Using LPM3 and MCLK for Baud Rate Generation

A solution using the MCLK for baud rate generation is provided in software listing *tc_uart2.asm,* with the MSP430 normally running in LPM3 and requiring a 9,600-baud UART. It is assumed that a 32,768-Hz watch crystal is used for XTAL, and that the default MCLK is 1,048,576-Hz. In this example, mainloop waits in LPM3 and the UART is ready to receive at 9,600 baud. What is different about this example is the initialization of the UART in the receive mode. Prior to entering the LPM3, when the RX_Ready subroutine is called to prepare the UART, P0.1 is configured as a pin-interrupt source—not as the 8-bit T/C overflow. This ensures that if the MCLK (DCO) is off (which occurs during LPM3), it will be restarted on a P0.1 edge, and the 8-bit T/C, which is clocked by MCLK, starts. Any accepted Interrupt starts the DCO. The edge-detect logic is still used (TCRXACT in TCCTL), so the 8-bit TC continues to start automatically on the falling edge of P0.1. After the first interrupt on P0.1, with the 8-bit T/C now counting, the P0.1 / 8-bit T/C interrupt source is reconfigured to that of the 8-bit T/C overflow, and the MCLK is set to remain on for the remainder of the received character. A compensation of about 6 µs is factored into bitime1_5 (timed from start-bit edge to the middle of the first data bit) to allow for DCO start. The subsequent data bits are hardware-latched as before.

4.2 Using LPM3 and ACLK for Baud-Rate Generation

In this application the MSP430x3xx runs normally in LPM3 and the ACLK (32,768-Hz watch crystal) is used as the clock source for the 8-bit T/C. If the LPM3 ACLK remains active, no software modifications are required for start bit detect. The 8-bit TC edge-detect logic on P0.1 automatically starts the 8-bit TC with ACLK as the timer clock. With the timer clocked by a relatively-slow clock, only low baud rates are possible (less than 4,800) to prevent bit timing errors. A solution for a 1,200-baud UART is provided in software listing *tc_uart4.asm*. For illustrative purposes, this example uses RAM locations for RXTXData and BitCnt, as opposed to the CPU registers used in the previous examples. An additional subroutine is also provided which sends a *Ready>* prompt to the PC, replacing the **>** used in the other examples. Sending *Ready>* illustrates a method of sending a sting from a buffer.

5 Summary

Even though the 8-bit T/C is not a dedicated serial port, it is easy to emulate such a function without excessively burdening the CPU. The unique DCO and MSP430x3xx FLL provide a high-speed stable-clock source usable for serial communication even when only a 32,768-Hz XTAL is used in the application. The 8-bit T/C has powerful hardware features that enable a UART and other functions not possible with simple timers. With true asynchronous capability, modern event-driven and multitasking-programming techniques can be exercised with the MSP430. External and internal events steer program flow reducing CPU overhead.

6 References

1. MSP430x31x data sheet, 1999, literature number SLAS165C
2. MSP430x32x data sheet, 1999, literature number SLAS219A
3. MSP430x33x data sheet, 1999, literature number SLAS163
4. *MSP430 Family Architecture and Module Library*, 1996, literature number SLAUE10B

Appendix A Software Listing tc_uart1.asm

```
            .include     ”STD_DEF.ASM”
;*******************************************************************
;   MSP430 DEMONSTRATION PROGRAM
;   9,600-baud UART Function from LPM0 using the 8-bit T/C and MCLK
;
;           -------------------- ----
;          |     MSP430x3x      |  32k XTAL
;          |--------------------|----
;      <-- | TX/P0.2 | 8bit T/C |
;      --> | RX/P0.1 |          |
;          |--------------------|
;
;   Description: This program runs normall in LPM0 and demonstrating a half
;   duplex UART function using 8-bit T/C.  ”>”  prompt is sent to the PC. The
;   MSP430x3xx waits in LPM3 until a character is received from the PC which
;   reactivates mainloop and the same character is echoed back to PC.
;   Program repeats.  MCLK used for baud generation.
;   [] This Program can be tested using 8N1 no flow control.
;
;   Conditions for 9,600 Baud UART with MCLK @ 1,048,576MHz
Bitime      .equ    0100h - 109     ; 104 us per bit  (104.2us actual)
Bitime1_5   .equ    0100h - 164     ; 1.5 bit length
TXLoad      .equ    0BAh            ; TCTXD=TCISCTL=TCTXEN=TCENCNT=1 MCLK
RXLoad      .equ    0B6h            ; TCTXD=TCISCTL=TCTXEN=TCRXACT=1 MCLK
;
;   M.Buccini
;   Americans Sales and Marketing
;   Texas Instruments, Inc
;   September 1999
;*******************************************************************
RAM_orig    .set    00200h          ; RAM Start address
ROM_orig    .set    0C000h          ; x3x5 ROM Start
Stack       .set    00400h          ; x3x5 TO of RAM Stackpointer
I_vectors   .set    0FFFFh          ; Interrupt vectors
Main        .equ    ROM_orig        ; Program Start
;
;   CPU Registers Used
RXTXData    .equ    R5              ; Register for RX / TX UART Data
BitCnt      .equ    R6              ; Register used to count UART bits
;-------------------------------------------------------------------
            .sect ”MAIN”,Main
;-------------------------------------------------------------------
RESET       mov     #Stack,SP       ; Initialize Stackpointer
            call    #Init_Sys       ; Setup Peripherals
;           Mainloop of Program
Mainloop    mov.b   #”>”,RXTXData   ; Load ”>” to buffer
            call    #TX_Byte        ; TX ”>”
            call    #RX_Ready       ; UART ready to RX one Byte
            bis     #LPM0,SR        ; Enter LPM0 Until Byte RXed
```

```
            call    #TX_Byte            ; TX Back RXed Byte Received
            jmp     Mainloop            ;
                                        ;
;-------------------------------------------------------------------------
Init_Sys    ; Subroutine: Setup MSP430 Peripherals For Operation
;-------------------------------------------------------------------------
SetupWDT    mov     #WDTPW+WDTHOLD,&WDTCTL   ; Stop WDT
Delay       mov     #0FFFFh,R15         ; SW Delay to allow FLL Lock
Delay1      dec     R15                 ;
            jnz     Delay1              ;
            eint                        ; Enable Interrupts
            ret                         ; Return from Setup
;
;-------------------------------------------------------------------------
TX_Byte     ; Subroutine: Transmit One Byte from RXTXData Buffer.
;-------------------------------------------------------------------------
            mov     #TX_Count,BitCnt    ; TX_Count--> Branch Pointer
            mov.b   #Bitime,&TCPLD      ; Load Preload with Bitime
            mov.b   #TXLoad,&TCCTL      ; Prepare and start timer
            bic.b   #P0IFG_1,&IFG1      ; Clear any erroneous interrupt flag
            bis.b   #P0IE_1,&IE1        ; Enable TC/P0.1 interrupt
TX_Wait     cmp     #TX_End+2,BitCnt    ; Wait for TX byte completion
            jne     TX_Wait             ;
            ret                         ; Return
                                        ;
;-------------------------------------------------------------------------
RX_Ready     ; Subroutine: Receive One Byte into RXTXData Buffer.
;-------------------------------------------------------------------------
            mov     #RX_Count,BitCnt    ; RX_Count--> Branch Pointer
            mov.b   #Bitime1_5,&TCPLD   ; Preload with 1.5 bit lengths
            mov.b   #0,&TCDAT           ; Force Bitime1_5 into timer
            mov.b   #Bitime,&TCPLD      ; Preload with Bitime, for data bit
            mov.b   #RXLoad,&TCCTL      ; Prepare for Start Bit
            bis.b   #P0IES_1,&P0IES     ; High/low edge for P0.1 interrupt
            bic.b   #P0IFG_1,&IFG1      ; Clear any erroneous interrupt flag
            bis.b   #P0IE_1,&IE1        ; Enable TC/P0.1 interrupt
            ret                         ;
;-------------------------------------------------------------------------
UART_Isr    ;  UART/P0.1 ISR
;-------------------------------------------------------------------------
            mov     @BitCnt+,PC         ; Branch To Routine
;
            .even                       ;
RX_Count    .word   RX_Bit              ; RX First Data Bit
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
```

```
            .word   RX_Comp             ; RX Complete, Process RXed Data
TX_Count    .word   TX_Space            ; TX Start Bit= Space
            .word   TX_Bit              ; TX First Data Bit
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Mark             ; TX Stop Bit= Mark
TX_End      .word   TX_Comp             ; TX Complete and Complete
;
TX_Bit      rra     RXTXData            ; TX ISR, LSB is shifted to carry
            jnc     TX_Space            ; Jump bit = 0TX_Mark
TX_Mark     bis.b   #TCTXD,&TCCTL       ; Bit= 1, set TCTXD in TCCTL
            reti
TX_Space    bic.b   #TCTXD,&TCCTL       ; Bit= 0, reset TCTXD in TCCTL
            reti
TX_Comp     bic.b   #TCENCNT,&TCCTL     ; Stop Timer
            bic.b   #P0IE_1,&IE1        ; Disable TC/P0.1 interrupt
            reti                        ;
RX_Bit      bit.b   #TCRXD,&TCCTL       ; RX ISR, Bit from TCRXD -> Carry
            rrc.b   RXTXData            ; Carry -> RXTXData
            reti                        ;
;       >>> Decode Received Byte Here <<<
RX_Comp     bic.b   #TCENCNT,&TCCTL     ; Stop Timer
            bic.b   #P0IE_1,&IE1        ; Disable TC/P0.1 interrupt
            mov     #GIE,0(SP)          ; Decode Byte= Active in Mainloop
            reti                        ;
;-----------------------------------------------------------------------
        .sect   ”Int_Vect”,I_vectors-31
;-----------------------------------------------------------------------
        .word   RESET                   ; Port0, bit 2 to bit 7
        .word   RESET                   ; Basic Timer
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; Timer Port
        .word   RESET                   ; EOC from ADC
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; Watchdog Timer, Timer mode
        .word   RESET                   ; no source
        .word   UART_Isr                ; P0.1 or 8-bit T/C
        .word   RESET                   ; P0.0
        .word   RESET                   ; NMI, Osc. fault
        .word   RESET                   ; POR, ext. Reset, Watchdog
```

Appendix B Software Listing tc_uart2.asm

```
            .include     "STD_DEF.ASM"
;*******************************************************************************
;   MSP430 DEMONSTRATION PROGRAM
;   9,600-baud UART Function from LPM3 using the 8-bit T/C and MCLK
;
;           --------------------- ----
;          |     MSP430x3x      |  32k XTAL
;          |--------------------|----
;      <-- | TX/P0.2 | 8bit T/C |
;      --> | RX/P0.1 |          |
;          |--------------------|
;
;   Description: This program runs normall in LPM3 and demonstrating a half
;   duplex UART function using 8-bit T/C.  ">"  prompt is sent to the PC. The
;   MSP430x3xx waits in LPM3 until a character is received from the PC which
;   reactivates mainloop and the same character is echoed back to PC.
;   Program repeats.  MCLK used for baud generation.
;   [] This Program can be tested using 8N1 no flow control.
;
;   Conditions for 9,600 Baud UART with MCLK @ 32x32768= 1,048,576
Bitime      .equ    0100h - 109       ; 104 us per bit  (104.2us actual)
Bitime1_5   .equ    0100h - (164-10)  ; 1.5 bit lengths - DCO start
TXLoad      .equ    0BAh              ; TCTXD=TCISCTL=TCTXEN=TCENCNT=1 MCLK
RXLoad      .equ    0B6h              ; TCTXD=TCISCTL=TCTXEN=TCRXACT=1 MCLK
RXLoadX     .equ    096h              ; TCTXD=TCTXEN=TCRXACT=1 MCLK
;
;   M.Buccini
;   Americans Sales and Marketing
;   Texas Instruments, Inc
;   September 1999
;*******************************************************************************
RAM_orig    .set    00200h            ; RAM Start address
ROM_orig    .set    0C000h            ; x3x5 ROM Start
Stack       .set    00400h            ; x3x5 TO of RAM Stackpointer
I_vectors   .set    0FFFFh            ; Interrupt vectors
Main        .equ    ROM_orig          ; Program Start
;
;   CPU Registers Used
RXTXData    .equ    R5                ; Register for RX / TX UART Data
BitCnt      .equ    R6                ; Register used to count UART bits
;-------------------------------------------------------------------------------
            .sect "MAIN",Main
;-------------------------------------------------------------------------------
RESET       mov     #Stack,SP         ; Initialize Stackpointer
            call    #Init_Sys         ; Setup Peripherals
;           Mainloop of Program
Mainloop    mov.b   #">",RXTXData     ; Load ">" to buffer
            call    #TX_Byte          ; TX ">"
```

```
            call    #RX_Ready           ; UART ready to RX one Byte
            bis     #LPM3,SR            ; Enter LPM3 Until Byte RXed
            call    #TX_Byte            ; TX Back RXed Byte Received
            jmp     Mainloop            ;
                                        ;
;-------------------------------------------------------------------------
Init_Sys    ; Subroutine: Setup MSP430 Peripherals For Operation
;-------------------------------------------------------------------------
SetupWDT    mov     #WDTPW+WDTHOLD,&WDTCTL   ; Stop WDT
Delay       mov     #0FFFFh,R15         ; SW Delay to allow FLL Lock
Delay1      dec     R15                 ;
            jnz     Delay1              ;
            eint                        ; Enable Interrupts
            ret                         ; Return from Setup
;
;-------------------------------------------------------------------------
TX_Byte     ; Subroutine: Transmit One Byte from RXTXData Buffer.
;-------------------------------------------------------------------------
            mov     #TX_Count,BitCnt    ; TX_Count--> Branch Pointer
            mov.b   #Bitime,&TCPLD      ; Load Preload with Bitime
            mov.b   #TXLoad,&TCCTL      ; Prepare and start timer
            bic.b   #P0IFG_1,&IFG1      ; Clear any erroneous interrupt flag
            bis.b   #P0IE_1,&IE1        ; Enable TC/P0.1 interrupt
TX_Wait     cmp     #TX_End+2,BitCnt    ; Wait for TX byte completion
            jne     TX_Wait             ;
            ret                         ; Return
                                        ;
;-------------------------------------------------------------------------
RX_Ready     ; Subroutine: Receive One Byte into RXTXData Buffer.
;-------------------------------------------------------------------------
            mov     #RX_Count,BitCnt    ; RX_Count--> Branch Pointer
            mov.b   #Bitime1_5,&TCPLD   ; Preload with 1.5 bit lengths
            mov.b   #0,&TCDAT           ; Force Bitime1_5 into timer
            mov.b   #Bitime,&TCPLD      ; Preload with Bitime, for data bit
            mov.b   #RXLoadX,&TCCTL     ; // LPM3 Prepare for Start Bit //
            bis.b   #P0IES_1,&P0IES     ; High/low edge for P0.1 interrupt
            bic.b   #P0IFG_1,&IFG1      ; Clear any erroneous interrupt flag
            bis.b   #P0IE_1,&IE1        ; Enable TC/P0.1 interrupt
            ret                         ;
                                        ;
;-------------------------------------------------------------------------
UART_Isr    ;  UART/P0.1 ISR
;-------------------------------------------------------------------------
            mov     @BitCnt+,PC         ; Branch To Routine
;
            .even                       ;
RX_Count    .word   RX_Edge             ; Start Bit Edge
            .word   RX_Bit              ; RX First Data Bit
            .word   RX_Bit              ;
```

```
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Bit              ;
            .word   RX_Comp             ; RX Complete, Process RXed Data
TX_Count    .word   TX_Space            ; TX Start Bit= Space
            .word   TX_Bit              ; TX First Data Bit
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Bit              ;
            .word   TX_Mark             ; TX Stop Bit= Mark
TX_End      .word   TX_Comp             ; TX Complete and Complete
;
TX_Bit      rra     RXTXData            ; TX ISR, LSB is shifted to carry
            jnc     TX_Space            ; Jump bit = 0TX_Mark
TX_Mark     bis.b   #TCTXD,&TCCTL       ; Bit= 1, set TCTXD in TCCTL
            reti
TX_Space    bic.b   #TCTXD,&TCCTL       ; Bit= 0, reset TCTXD in TCCTL
            reti
TX_Comp     bic.b   #TCENCNT,&TCCTL     ; Stop Timer
            bic.b   #P0IE_1,&IE1        ; Disable TC/P0.1 interrupt
            reti                        ;
RX_Edge     bis.b   #TCISCTL,&TCCTL     ; 8-it T/C Now Interrupt Source
            bic     #SCG1+SCG0,0(SP)    ; DCO and FLL Now on after reti
            reti
RX_Bit      bit.b   #TCRXD,&TCCTL       ; RX ISR, Bit from TCRXD -> Carry
            rrc.b   RXTXData            ; Carry -> RXTXData
            reti                        ;
;       >>> Decode Received Byte Here <<<
RX_Comp     bic.b   #TCENCNT,&TCCTL     ; Stop Timer
            bic.b   #P0IE_1,&IE1        ; Disable TC/P0.1 interrupt
            mov     #GIE,0(SP)          ; Decode Byte= Active in Mainloop
RXTX_Next   reti                        ;
                                        ;
;-------------------------------------------------------------------------------
        .sect   "Int_Vect",I_vectors-31
;-------------------------------------------------------------------------------
        .word   RESET                   ; Port0, bit 2 to bit 7
        .word   RESET                   ; Basic Timer
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; Timer Port
        .word   RESET                   ; EOC from ADC
```

```
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; Watchdog Timer, Timer mode
        .word   RESET                   ; no source
        .word   UART_Isr                ; P0.1 or 8-bit T/C
        .word   RESET                   ; P0.0
        .word   RESET                   ; NMI, Osc. fault
        .word   RESET                   ; POR, ext. Reset, Watchdog
```

Appendix C Software Listing tc_uart4.asm

```
                    .include     ”STD_DEF.ASM” ; listed in the listing file or
;***************************************************************************
;   MSP430 DEMONSTRATION PROGRAM
;   1200-baud UART Function from LPM3 using the 8-bit T/C and ACLK
;
;           -------------------- ----
;          |     MSP430x3x      |  32k XTAL
;          |--------------------|----
;      <-- | TX/P0.2 | 8bit T/C |
;      --> | RX/P0.1 |          |
;          |--------------------|
;
;   Description: This program runs normall in LPM3 and demonstrating a half
;   duplex UART function using 8-bit T/C.  ”>”  prompt is sent to the PC. The
;   MSP430x3xx waits in LPM3 until a character is received from the PC which
;   reactivates mainloop and the same character is echoed back to PC.
;   Program repeats.  MCLK used for baud generation.
;   [] This Program can be tested using 8N1 no flow control.
;
;           Conditions for 1200 Baud UART with ACLK @ 32768
Bitime      .equ    0100h - 27      ; 823.5us per bit (833.3us actual)
Bitime1_5   .equ    0100h - 43      ; 1.5 bit lengths
TXLoad      .equ    07Ah            ; TCTXD=TCISCTL=TCTXEN=TCENCNT=1 ACLK
RXLoad      .equ    076h            ; TCTXD=TCISCTL=TCTXEN=TCRXACT=1 ACLK
;
;   M.Buccini
;   Americans Sales and Marketing
;   Texas Instruments, Inc
;   September 1999
;***************************************************************************
RAM_orig    .set    00200h          ; RAM Start address
ROM_orig    .set    0C000h          ; x3x5 ROM Start
Stack       .set    00400h          ; x3x5 TO of RAM Stackpointer
I_vectors   .set    0FFFFh          ; Interrupt vectors
Main        .equ    ROM_orig        ; Program Start
;
;   RAM Registers Used
RXTXData    .equ    0200h           ; Register for RX or TX UART Data
BitCnt      .equ    0202h           ; Register used to count UART bits
Pointer     .equ    R7              ; Pointer used for table processing
LF          .equ    0ah             ; ASCII Line Feed
CR          .equ    0dh             ; ASCII Carriage Return
;---------------------------------------------------------------------------
            .sect ”MAIN”,Main
;---------------------------------------------------------------------------
RESET       mov     #Stack,SP       ; Initialize Stackpointer
            call    #Init_Sys       ; Setup Peripherals
;           Mainloop of Program
```

```
Mainloop    call    #TX_Prompt          ; TX ”Ready>” Prompt
            call    #RX_Ready           ; UART ready to RX one Byte
            bis     #LPM3,SR            ; Enter LPM0 Until Byte RXed
            call    #TX_Byte            ; TX Back RXed Byte Received
            jmp     Mainloop            ;
                                        ;
;-------------------------------------------------------------------------
Init_Sys    ; Subroutine: Setup MSP430 Peripherals For Operation
;-------------------------------------------------------------------------
SetupWDT    mov     #WDTPW+WDTHOLD,&WDTCTL   ; Stop WDT
Delay       mov     #0FFFFh,R15         ; SW Delay to allow FLL Lock
Delay1      dec     R15                 ;
            jnz     Delay1              ;
            eint                        ; Enable Interrupts
            ret                         ; Return from Setup
;
;-------------------------------------------------------------------------
TX_Prompt    ;   Transmit ”Ready>” Prompt String
;-------------------------------------------------------------------------
            mov     #Buffer,Pointer     ; Pointer Points Buffer String
Prompt1     mov.b   @Pointer+,RXTXData  ; Move String Byte TO UART
            call    #TX_Byte            ; Send Byte
            tst.b   0(Pointer)          ; End of String Deliminator ”0”
            jnz     Prompt1             ; If ”0”, String complete
            ret                         ; Done
Buffer      .string CR,LF, ”Ready>”
            .byte   0
;
;-------------------------------------------------------------------------
TX_Byte     ; Subroutine: Transmit One Byte from RXTXData Buffer.
;-------------------------------------------------------------------------
            add     #0100h,&RXTXData    ; Add Mark stop Bit to RXTXData
            rla     &RXTXData           ; Add Space start Bit to RXTXData
            mov     #10,&BitCnt         ; Load Bit Counter, 8 data + ST +SP
            mov.b   #TXLoad,&TCCTL      ; Prepare and start timer
            bic.b   #P0IFG_1,&IFG1      ; Clear any erroneous interrupt flag
            bis.b   #P0IE_1,&IE1        ; Enable TC/P0.1 interrupt
TX_Wait     tst     &BitCnt             ; Wait for TX Completion
            jnz     TX_Wait             ;
            ret                         ;
                                        ;
;-------------------------------------------------------------------------
RX_Ready     ; Subroutine: Receive One Byte into RXTXData Buffer.
;-------------------------------------------------------------------------
            mov.b   #Bitime1_5,&TCPLD   ; Preload with 1.5 bit lengths
            mov.b   #0,&TCDAT           ; Force Bitime1_5 into timer
            mov.b   #Bitime,&TCPLD      ; Preload with Bitime, for data bit
            mov     #08,&BitCnt         ; Load Bit Counter, 8 data bits
            mov.b   #RXLoad,&TCCTL      ; Prepare for Start Bit
```

```
            bis.b   #P0IES_1,&P0IES     ; High/low edge for P0.1 interrupt
            bic.b   #P0IFG_1,&IFG1      ; Clear any erroneous interrupt flag
            bis.b   #P0IE_1,&IE1        ; Enable TC/P0.1 interrupt
            ret                         ;
                                        ;
;-------------------------------------------------------------------------
UART_Isr    ;  UART/P0.1 ISR
;-------------------------------------------------------------------------
            bit.b   #TCRXACT,&TCCTL     ; Receive or transmit?
            jnz     RX_Bit              ; Goto RX
UART_TX     rra     &RXTXData           ; LSB is shifted to carry
            jnc     TX_Space            ; Jump bit = 0
TX_Mark     bis.b   #TCTXD,&TCCTL       ; Bit= 1, set TCTXD in TCCTL
            jmp     TX_Test             ; Next bit
TX_Space    bic.b   #TCTXD,&TCCTL       ; Bit= 0, reset TCTXD in TCCTL
TX_Test     dec     &BitCnt             ; All bits sent (or received)?
            jnz     RXTX_Next           ; Next bit?
TX_Comp     bic.b   #TCENCNT,&TCCTL     ; Stop Timer
            bic.b   #P0IE_1,&IE1        ; Disable TC/P0.1 interrupt
TX_Next     reti                        ;
RX_Bit      bit.b   #TCRXD,&TCCTL       ; RX ISR, Bit from TCRXD -> Carry
            rrc.b   RXTXData            ; Carry -> RXTXData
RX_Test     dec     &BitCnt             ; All bits sent (or received)?
            jnz     RXTX_Next           ; Next bit?
;       >>> Decode Received Byte Here <<<
RX_Comp     bic.b   #TCENCNT,&TCCTL     ; Stop Timer
            bic.b   #P0IE_1,&IE1        ; Disable TC/P0.1 interrupt
            mov     #GIE,0(SP)          ; Decode Byte= Active in Mainloop
RXTX_Next   reti                        ;
                                        ;
;-------------------------------------------------------------------------
        .sect   ”Int_Vect”,I_vectors-31
;-------------------------------------------------------------------------
        .word   RESET                   ; Port0, bit 2 to bit 7
        .word   RESET                   ; Basic Timer
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; Timer Port
        .word   RESET                   ; EOC from ADC
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; no source
        .word   RESET                   ; Watchdog Timer, Timer mode
        .word   RESET                   ; no source
        .word   UART_Isr                ; P0.1 or 8-bit T/C
        .word   RESET                   ; P0.0
        .word   RESET                   ; NMI, Osc. fault
        .word   RESET                   ; POR, ext. Reset, Watchdog
```

IMPORTANT NOTICE

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

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

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

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

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