TEXAS INSTRUMENTS

MSP430 Universal Synchronous Asynchronous Receive/Transmit Communication Interface

Application Report

April 1999

Mixed Signal Products

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.

Contents

List of Figures

List of Tables

MSP430 Universal Synchronous Asynchronous Receive/Transmit Communication Interface

Lutz Bierl

ABSTRACT

This application report gives a short overview for the use of the MSP430 universal synchronous, asynchronous receive/transmit communication interface (USART) as an RS232 interface, also called a serial-controller interface (SCI). Tested software examples, with and without the use of the interrupt capability, are given for the transmission and the reception of UART (universal asynchronous receive/transmit) signals. Full duplex mode is used for all examples running in active mode and low power mode 3 (LPM3).

1 Introduction

The universal synchronous/asynchronous receive/transmit communication interface of the MSP430 family can operate in two different modes: synchronous and asynchronous. This application report describes the software routines used with the asynchronous mode (SCI, RS232). A second report will handle the synchronous mode (serial protocol interface SPI).

> **NOTE:** Reading the data book *MSP430 Family Architecture Guide and Module Library* is recommended. It complements the information contained in this application report.

> **NOTE:** The examples and hardware definitions use MSP430x33x addresses. Future MSP430 family members may have different hardware addresses—especially for the I/O ports used.

The hardware features of the USART module greatly exceed the capabilities illustrated in the examples included in this application report. This report is intended as a fast way to get the USART running in the UART mode, with or without the interrupt capability. Features frequently used are included in the examples.

Figure 1 shows the block diagram of the MSP430 USART module.

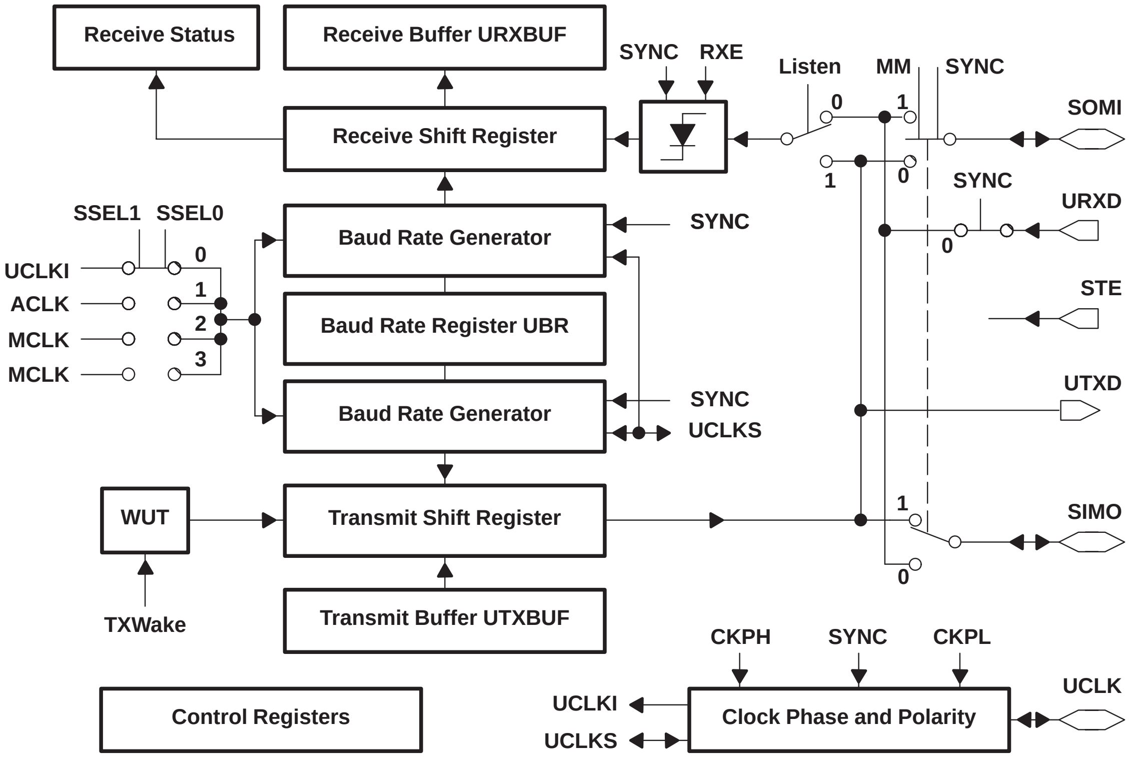

Figure 1. MSP430 USART Module

Figure 2 shows the situation when the USART is switched to the UART mode by setting the SYNC bit UCTL.2 to zero

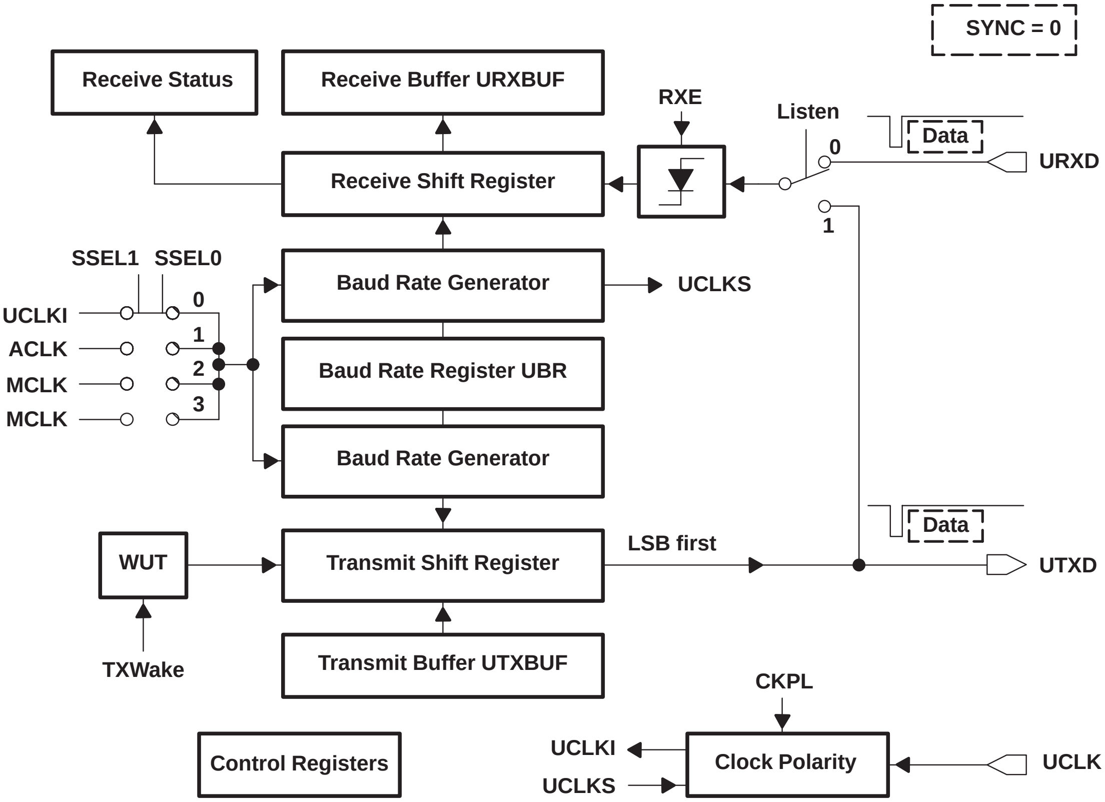

Figure 2. USART Switched to the UART Mode

1.1 Attributes of the MSP430 UART

The following is a short overview of the USART running in the UART mode:

- Selectable seven and eight-bit data lengths
- The error detection for the receive path is as follows:
  - Frame error. The stop bits have space potential.
  - Parity error. Parity is enabled and the parity bit has the wrong value.
  - Overrun error. The next character is read in before the last one is read out by the software.
  - Break detect. The URXD pin has low potential for more than 10 bits.
- Baud-rate generation made possible by 32-kHz crystal due to the modulation register
- Interrupt-driven transmit and receive functions
- Two independent interrupt vectors: one for transmission, one for reception
- Full functionality during LPM3
- End-of-frame flag usable with interrupt or polling

1.2 Data Format

The data format used is RS232. Figure 3 shows how this format is seen at the MSP430 ports (URXD and UTXD), and Figure 4 shows how the format is defined on the transmission line. The format shown in Figures 3 and 4 has:

- Seven data bits. The least-significant bit follows the start bit.
- Parity enabled. The parity bit follows the most-significant bit of the data.
- No address bit. This is the normal case.
- Two stop bits

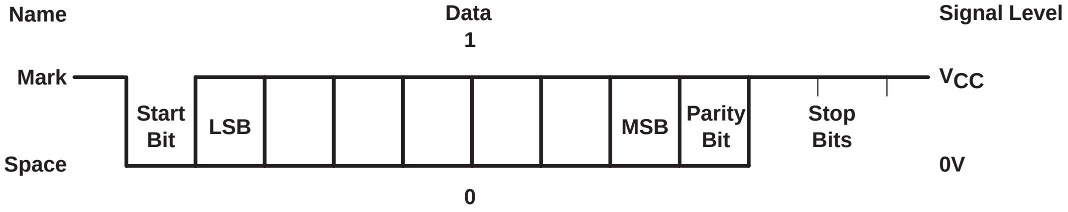

Figure 3. RS232 Format (Levels at the MSP430)

The signal on the transmission line has the inverted state when observed at the MSP430 ports, and different voltage potentials. This is illustrated in Figure 4.

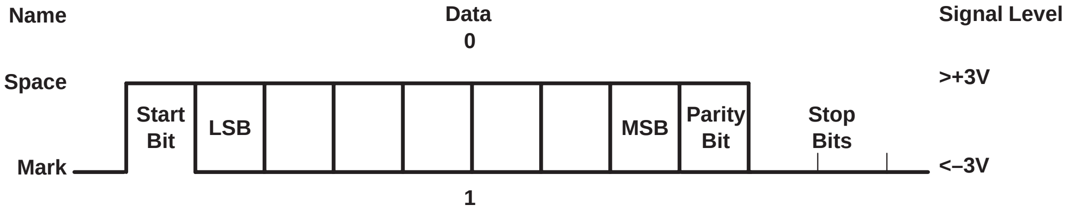

Figure 4. RS232 Format (Levels on the Transmission Line)

1.3 UART Hardware Registers

The USART is controlled by seven control registers and one read-only register. All of them are 8-bit registers and consequently should only be accessed with byte instructions. Figure 5 gives an overview of these eight registers including the names, assembler mnemonics, hardware addresses, and their initial states. The register and bit definitions are found in Appendix A.

Register Name	Mnemonic	Register Access	Address	Initial State
USART Control Register	UCTL	Read/Write	070h	See below
Transmit Control Register	UTCTL	Read/Write	071h	See below
Receive Control Register	URCTL	Read/Write	072h	See below
Modulation Control Register	UMCTL	Read/Write	073h	unchanged
Baud-Rate Register 0	UBR0	Read/Write	074h	unchanged
Baud-Rate Register 1	UBR1	Read/Write	075h	unchanged
Receive Buffer	URXBUF	Read Only	076h	unchanged
Transmit Buffer	UTXBUF	Read/Write	077h	unchanged

	7							0
UCTL 070h	PENA	PEV	SP	CHAR	Listen	SYNC	MM	SWRST
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

	7							0
UBR0 074h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^1
	rw	rw	rw	rw	rw	rw	rw	rw

	7							0
UTCTL 071	un- used	CKPL	SSEL 1	SSEL0	URXSE	TXWake	unused	TXEPT
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

	7							0
UBR1 075h	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
	rw	rw	rw	rw	rw	rw	rw	rw

	7							0
URCTL 072h	FE	PE	OE	BRK	URXEIE	URXWIE	RXWake	RXERR
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

	7							0
UMCTL 073h	m7	m6	m5	m4	m3	m2	m1	m0
	rw	rw	rw	rw	rw	rw	rw	rw

	7							0
URXBUF 076h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	r	r	r	r	r	r	r	r

	7							0
UTXBUF 077h	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	rw	rw	rw	rw	rw	rw	rw	rw

Figure 5. USART Control Registers Used in the UART Mode

2 Baud Rate Generation

It is easy to generate the desired baud rate from a relatively high frequency (1 MHz to 5 MHz): the resulting baud rate error is small due to the large integer-part of the quotient compared to the truncated fractional-part. This is not the case when the time base used is a 32-kHz crystal, since the error due to the truncated fractional-part of the quotient becomes large and leads to the loss of synchrony at the trailing bits of the frame. For this reason, the MSP430 USART uses a correction to keep the baud rate error small. The modulation register UMCTL contains 8 bits of information to correct the baud rate of the received or transmitted UART signal. These bits determine the use of the predivider information contained in the two baud-rate registers UBR0 and UBR1:

- A *Zero* bit in the UMCTL register means that the information contained in UBR1/UBR0 is used as is.
- A *One* bit means that the 16-bit content of UBR1/UBR0 is incremented by one before it is used. The content of UBR1/UBR0 is not changed.

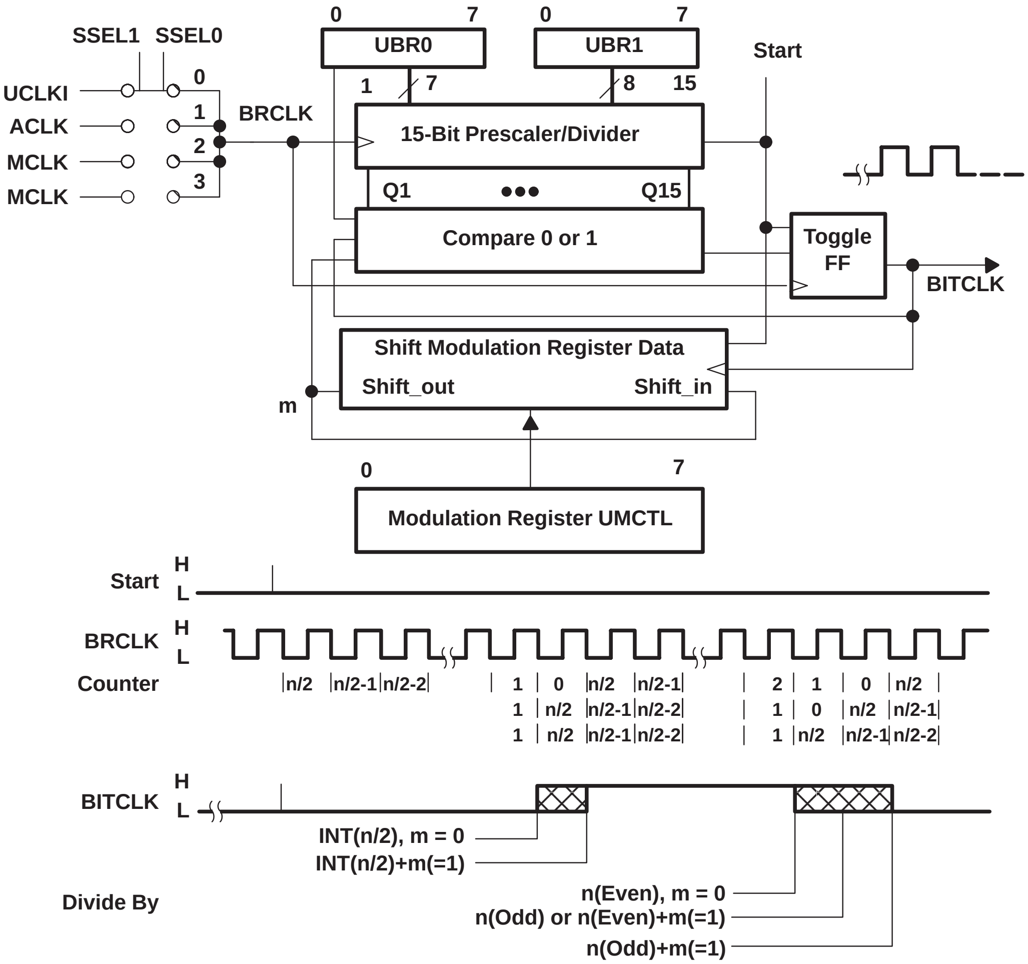

Figure 6. Baud Rate Generator

The LSB (m0) of register UMCTL is used for the start bit, the next bit (m1) for the LSB of the data, and so on. After using bit m7, the bit sequence m0 thru m7 is repeated. See Figure 7 for a graphic explanation.

EXAMPLE: a baud rate of 4800 baud is required with a crystal frequency of 32,768 Hz. This is necessary because the UART also has to run during low power mode 3. With only the ACLK available, the theoretical division factor—the truncated value is the content of baud-rate register UBR (UBR1/UBR0)—is:

$$UBR = \frac{32768}{4800} = 6.82667$$

This means that the baud-rate register UBR1 (MSBs) is loaded with zero, and the UBR0 register contains a 6. To get a rough estimate of the 8-bit modulation register UMCTL, the fractional part 0.826667 is multiplied by 8 (the number of bits in register UMCTL):

$$UMCTL = 0.82667 \times 8 = 6.613$$

The rounded result 7 is the number of *ones* to be placed into the modulation register UMCTL. The corrected baud rate with the UMCTL register containing 7 ones is:

$$baud\ rate = \frac{32768}{\left(\frac{7 \times 7 + 1 \times 6}{8}\right)} = 4766.2545$$

This results in an average baud rate error of:

$$baud\ rate\ error = \frac{4766.2545 - 4800}{4800} \times 100 = -0.703\%$$

To get the best-fitting bit sequence for modulation register UMCTL, the following algorithm can be used: the fractional part of the theoretical division factor is summed up eight times; the actual m-bit is set if a carry to the integer part occurs, and is cleared otherwise. An example using the fraction 0.82667 previously calculated follows:

Fraction Addition	**Carry to next integer**	**UMCTL Bits**	
0.82667 + 0.82667 = 1.65333	Yes	m0	1
1.65333 + 0.82667 = 2.48000	Yes	m1	1
2.48000 + 0.82667 = 3.30667	Yes	m2	1
3.30667 + 0.82667 = 4.13333	Yes	m3	1
4.13333 + 0.82667 = 4.96000	No	m4	0
4.96000 + 0.82667 = 5.78667	Yes	m5	1
5.78667 + 0.82667 = 6.61333	Yes	m6	1
6.61333 + 0.82667 = 7.44000	Yes	m7	1

The result of the calculated bits m7...m0 is EFh (11101111b). Section 3.3.2 contains a software macro (CALC_UMCTL) that uses this algorithm to calculate the optimum value for the modulation register UMCTL for every combination of USART clock and desired baud rate. For the above example, the algorithm also finds EFh with its seven ones.

A second software macro (CALC_UBR) calculates the values of the two UBR registers.

EXAMPLE: Figure 7 presents an example using a 2400-baud rate generated with the ACLK frequency (32,768 Hz). The data format for Figure 7 is: eight data bits, parity enabled, no address bit, two stop bits.

Figure 7 shows three different frames:

- The upper frame is the correct one, with a bit-length of 13.65333 ACLK cycles (32,768/2400 = 13.65333).
- The middle frame uses a rough estimate, with 14 ACLK cycles for the bit length.
- The lower frame shows a corrected frame using the best fit (6Dh) for the modulation register.

It can be observed that the approximation with 14 ACLK cycles produces a cumulative error of more than 0.3 bit-lengths after the second stop bit. The error of the corrected frame is only 0.011 bit-lengths. The error of the crystal clock, not yet included, adds to the above error.

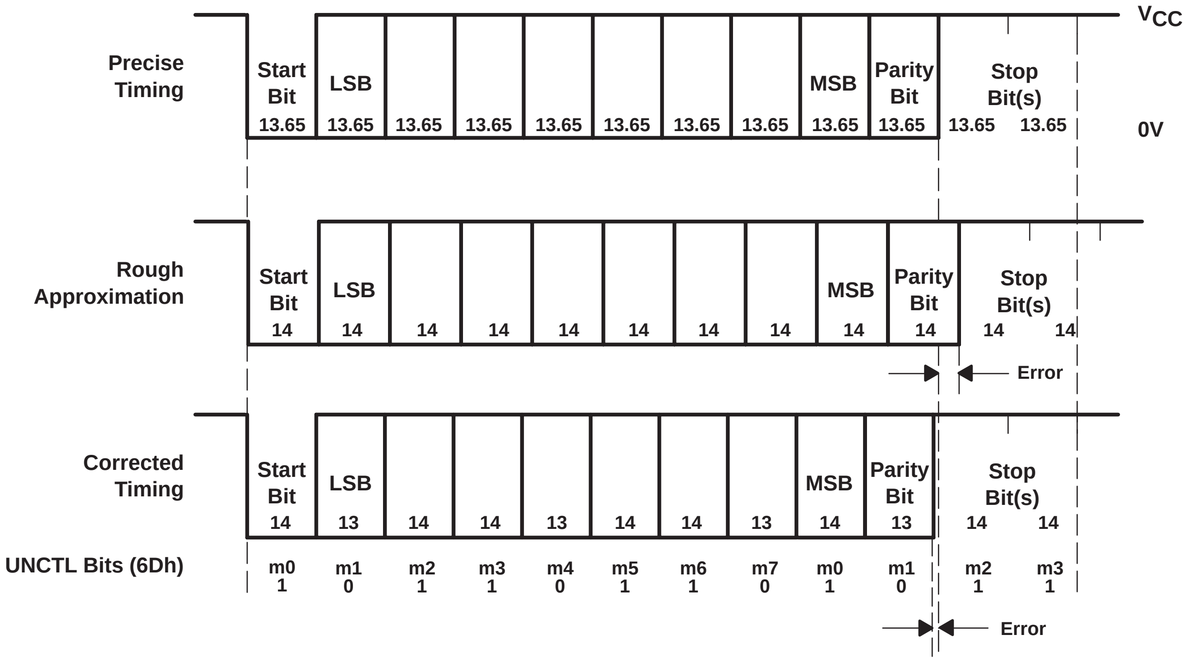

Figure 7. Function of the Baud Rate Correction

Tables 1 and 2 contain the average errors (full frame) for commonly used baud rates when using the described baud rate generation.

The software examples in section 3 contain MACROs that automatically insert the correct values in the UBR registers and in the modulation register UMCTL. The software MACROs, which do not need ROM or RAM, may be hidden in the listing by a .mnolist assembler directive.

2.1 Baud Rate Generation With the MCLK

Table 1 shows the optimum values for the UBR and UMCTL registers. The UART clock is the MCLK (1.048 MHz). The values for the UMCTL and UBR1/UBR0 registers are calculated by the software MACROs in section 3.3.2. The crystal error is not included.

Table 1 contains the following columns:

- **Baud Rate:** The baud rate for data exchange (transmit and receive use the same baud rate).
- **Division Factor:** The quotient UARTCLK/baud rate.
- **UBR1/UBR0 Content:** The truncated 16-bit hexadecimal result of the division factor (UARTCLK/baud rate). Its value is calculated by software macro CALC_UBR. The high byte is the UBR1 value, the low byte is the UBR0 value.
- **Calculated UMCTL Content:** The 8-bit result that best fits the modulation register. It is calculated by the software macro CALC_UMCTL.
- **Used Fraction:** The number of ones in the modulation register divided by eight. It is an approximation to the truncated fractional-part of the division factor.
- **Mean Error:** The resulting error of a complete character, caused by the approximation to the division factor.

Table 1. Content of Baud Rate Registers UBR (MCLK = 1.048 MHz)

BAUD RATE	DIVISION FACTOR	UBR1/UBR0 CONTENT	CALCULATED UMCTL CONTENT	USED FRACTION	MEAN ERROR [%]
110	9532.51	253Ch	55h	0.50	+0.000
300	3495.25	0DA7h	44h	0.25	0.000
600	1747.63	06D3h	6Dh	0.625	+0.000
1200	873.81	0369h	EFh	0.875	–0.007
2400	436.91	01B4h	FFh	1.00	–0.002
4800	218.45	00DAh	AAh	0.50	–0.023
9600	109.23	006Dh	88h	0.25	–0.018
19200	54.61	0036h	ADh	0.625	–0.027
38400	27.31	001Bh	24h	0.25	+0.220

2.2 Baud Rate Generation With the ACLK

With the relatively low ACLK frequency (32,768Hz), the importance of the modulation register UMCTL is much greater than with the normally high MCLK frequency used for the UART timing. Table 2 shows the optimum values for the UBR and UMCTL registers for commonly used baud rates generated with the ACLK (32,768Hz). The table values are calculated by the MACROs described in section 3.3.2. The crystal is assumed to have no frequency-error. The meaning of the table columns is explained in section 2.1.

Table 2. Content of Baud Rate Registers UBR (ACLK = 32,768 Hz)

BAUD RATE	DIVISION FACTOR	UBR1/UBR0 CONTENT	CALCULATED UMCTL CONTENT	USED FRACTION	MEAN ERROR [%]
110	297.8909	0129h	FFh	1.00	–0.04
300	109.2267	006Dh	88h	0.25	–0.02
600	54.6133	0036h	ADh	0.625	–0.02
1200	27.3067	001Bh	24h	0.25	+0.21
2400	13.6533	000Dh	6Dh	0.625	+0.21
4800	6.8267	0006h	EFh	0.875	–0.71
9600	3.4133	0003h	4Ah	0.375	+1.12
19200	1.7067	–			
38400	0.8533	–			

3 Software Routines

The following sections show proven software routines, subroutines, and software MACROs for the UART mode of the USART.

> **NOTE:** The program sequence for the initialization of the UART is important: as long as the SWRST bit (UCTL.0) is set, the receive and transmit control registers URCTL and UTCTL can not be initialized. The program sequences given in the software examples are recommended because they comply with this rule.
>
> While the SWRST bit is zero, the following control bits are held in the zero state: TXWAKE, RXERROR, RXWAKE, BRK, OE, FE, PE, URXIFG, URXIE, UTXIE.
>
> The following control bits are held in the one state: UTXIFG and TXEPT.

3.1 Non-Interrupt Processing

The simplest way to use the USART in UART mode is this: interrupt is not enabled, the software checks if it can output the next byte (UTXIFG = 1), and if a new character has been received (URXIFG = 1).

EXAMPLE: full duplex UART software code running without the use of the UART interrupt is shown. It is designed for:

- Baud rate: 1200 baud
- MCLK (1.048MHz) used as the UART clock
- Eight data bits
- Two stop bits
- Parity enabled with odd parity
- Receive of error-free characters only

```
STACK        .equ   0600h        ; Stack start address
;
; Definitions for the UART part: user defined
;
```

```
Baudr        .equ   1200         ; Baud rate is 1200 Baud
FLLMPY       .equ   32           ; FLL multiplier for 1,048MHz
UARTCLK      .equ   FLLMPY*32768 ; MCLK is used for UARTCLK
;
; The contents of the UMCTL and UBR registers are calculated.
; The two software macros do not use RAM or ROM, they only
; define the variables CUMCTL, CUBR1 and CUBR0 for the
; UART registers UMCTL, UBR1 and UBR0.
;
             CALC_UMCTL          ; Calc. Modulation Reg. content
             CALC_UBR            ; Calculate UBR1/UBR0 contents
;
             .text               ; Software start address
;
INIT         MOV    #STACK,SP    ; Initialize Stack Pointer
             CALL   #INITSR      ; Init. FLL and RAM
             ..                  ; Proceed with initialization
;
; Initialize the UART: odd parity, 8 data bits, 2 stop bits
; MCLK for UART clock
;
             MOV.B  #CUMCTL,&UMCTL      ; Modulation Register
             MOV.B  #CUBR0,&UBR0        ; Baud Rate Register low
             MOV.B  #CUBR1,&UBR1        ; Baud Rate Register high
             BIS.B  #URXD+UTXD,&P4SEL   ; Select RXD + TXD at Port4
             BIS.B  #UTXE+URXE,&ME2     ; Enable USART Modules
             MOV.B  #PENA+SP_+CHAR,&UCTL ; USART Control Register
             MOV.B  #SSEL1+SSEL0,&UTCTL ; Transmit Control Reg. MCLK
             MOV.B  #0,&URCTL           ; Receive Control Register
             ...                        ; Continue with initialization
;
MAINLOOP     ...                        ; Start Mainloop
;
; UART parts within the main loop.
; The software checks these two parts regularly.
; UART Receive part:
; check if a new character is received
; R7 contains the received information.
             BIT.B  #RXERR,&URCTL       ; Error during receive?
             JZ     L$3                 ; No
             ...                        ; Error handling
             BIC.B  #FE+PE+OE+BRK+RXERR,&URCTL ; Clear error flags
             JMP    L$2                 ; Continue in mainloop
```

```
;
L$3          BIT.B  #URXIFG,&IFG2       ; Character received?
             JZ  L$2                    ; No, proceed to mainloop
             MOV.B  &URXBUF,R7,         ; Yes, move character to R7
L$2          ...                        ; Continue in mainloop
;
; UART Transmit part:
; check if the next character can be transmitted.
; R6 contains information to be transmitted.
;
             BIT.B  #UTXIFG,&IFG2       ; Transmit buffer empty?
             JZ  L$1                    ; No, wait
             MOV.B  R6,&UTXBUF          ; Empty: move info to TX buffer
             MOV.B  src,R6              ; Next character to R6
L$1          ...                        ; Continue with mainloop
             BR  #MAINLOOP              ; End of mainloop
;
; Interrupt Vectors
;
             .sect  ”INITVEC”,0FFFEh    ; Reset Vector
             .word  INIT                ; Program Start Address
```

Only the following two source lines need to be modified if the previous software is to be used with the ACLK as the UART clock:

```
UARTCLK      .equ   32768               ; ACLK is used for UARTCLK
;
             MOV.B  #SSEL0,&UTCTL       ; Transmit Control Register ACLK
```

Macros CALC_UMCTL and CALC_UBRAll automatically make any other modifications necessary.

3.2 Interrupt Processing

This is the normal mode to use the UART. Interrupt is requested if the general interrupt enable bit GIE (SR.3) is set, and:

- A character is transmitted, and the transmit interrupt is enabled (IE2.1 = 1), or
- A character is received, and the receive interrupt is enabled (IE2.0 = 1)

NOTE: If an error occurred during the reception of a character, the error flags in the receive control register (PE, FE, BRK, and RXERR) must be reset within the UART interrupt handler. Otherwise the set-error flags will block the next interrupt. This does not apply to the overrun-error flag OE.

3.2.1 MCLK Used as UART Clock

The following example covers the use of the MCLK to generate the UART clock or external frequencies in the MCLK range (500 kHz to 3.8 MHz).

For high baud-rates—higher than 38,400 baud—dedicated CPU registers may be necessary to lower the interrupt overhead; time for saving and restoring the register is not required. The software example shown in section 3.2.2 uses dedicated registers.

EXAMPLE: a full-duplex UART software using the two UART interrupts is shown. It is designed for:

- Baud rate: 19200 baud
- The MCLK (3.144 MHz) used as the UART clock
- Seven data bits
- One stop bit
- Parity enabled with even-parity
- Receive of error-free characters only

Transmit Part: the start address xxxx is loaded into pointer TXPOI, and the number of characters to be output is loaded into character count TXCNT. The interrupt routine outputs the programmed character sequence starting at address xxxx.

Receive Part: the start address yyyy of a RAM buffer is loaded into pointer RCPOI, and the number of characters to be received is loaded into character count RCCNT. The interrupt routine receives the characters and stores them into the buffer. Only error-free characters are accepted.

```
STACK     .equ   0600h                 ; Stack start address
;
; Definitions for the UART part
;
Baudr     .equ   19200                 ; Baud rate is 19200 Baud
FLLMPY    .equ   96                    ; FLL multiplier for 3,144MHz
UARTCLK   .equ   FLLMPY*32768          ; MCLK is used for UARTCLK
;
          .even                        ; Word boundary
          .bss   TXPOI,2               ; Pointer to transmit buffer
          .bss   RCPOI,2               ; Pointer to receive buffer
          .bss   TXCNT,1               ; Counter/status for transmit
          .bss   RCCNT,1               ; Counter/status for receive
;
; The content for the UMCTL and UBR registers are calculated
; The two software macros do not use RAM or ROM
;
          CALC_UMCTL                   ; Calculate Mod. Reg. content
          CALC_UBR                     ; Calculate UBR1/UBR0 contents
;
          .text                        ; Software start address
;
INIT      MOV    #STACK,SP             ; Initialize Stack Pointer
```

```
          CALL   #INITSR                ; Init. FLL and RAM
          ...                           ; Proceed with initialization
;
; Initialize the UART: Even parity, 7 data bits, 1 stop bit
; MCLK for UART clock, only error-free characters to URXBUF
;
          MOV.B  #CUMCTL,&UMCTL         ; Modulation Register
          MOV.B  #CUBR0,&UBR0           ; Baud Rate Register low
          MOV.B  #CUBR1,&UBR1           ; Baud Rate Register high
          BIS.B  #URXD+UTXD,&P4SEL      ; Select RXD + TXD at Port4
          BIS.B  #UTXE+URXE,&ME2        ; Enable USART Modules
          MOV.B  #PENA+PEV,&UCTL        ; USART Control Register
          MOV.B  #SSEL1+SSEL0,&UTCTL    ; Transmit Control Reg. MCLK
          MOV.B  #0,&URCTL              ; Receive Control Register
          BIS.B  #UTXIE+URXIE,&IE2      ; Enable USART interrupts
          CLR.B  TXCNT                  ; Disable transmit
          CLR.B  RCCNT                  ; Disable receive
          ...                           ; Continue with initialization
          EINT      ; Enable interrupt
;
MAINLOOP  ...                           ; Start of Mainloop
;
; Preparation for reception of m bytes. The input
; buffer starts at address yyyy
;
          TST.B  RCCNT                  ; Data input completed?
          JNZ    L$1                    ; No, wait
          MOV    #yyyy,RCPOI            ; Buffer start address to RCPOI
          MOV.B  #m,RCCNT               ; Number of bytes to RCCNT
L$1       ...                           ; Continue in mainloop
;
; Stop the reception of data. The currently received character
; is input completely
;
          CLR.B  RCCNT                  ; Status to zero
          ...                           ; Continue
;
; Preparation for the transmission of n bytes starting at
; address xxxx. Check if last transmit operation
; is really completed.
;
          BIT.B  #TXEPT,&UTCTL          ; Transmit part ready?
          JZ     L$2                    ; No, buffers are not yet empty
```

```
;
       MOV.B  #n–1,TXCNT                  ; Ready, init. byte count
       MOV    #xxxx+1,TXPOI               ; Init. transmit buffer pointer
       MOV.B  xxxx,&UTXBUF                ; First info byte to TX buffer
L$2    ...                                ; Continue in background
;
; Stop the transmission of data. The currently sent character
; is transmitted completely
;
       CLR.B  TXCNT                       ; Status to zero
       ...
;
; Interrupt Handlers
; Interrupt handler for the UART Receive part.
;
RCINT  TST.B  RCCNT                       ; Reception allowed?
       JZ     RCRET                       ; No, status is 0
       BIT.B  #RXERR,&URCTL               ; Error during receive?
       JNZ RCERR                          ; Yes
       DEC.B  RCCNT                       ; No, Byte count –1
       PUSH   R5                          ; Save R5
       MOV    RCPOI,R5                    ; Pointer to buffer
       MOV.B  &URXBUF,0(R5)               ; Next byte to buffer
       INC    R5                          ; To next buffer byte
       MOV    R5,RCPOI                    ; Update pointer
       POP    R5                          ; Restore R5
RCRET  RETI
;
RCERR  ...                                ; Error handling
       BIC.B  #FE+PE+OE+BRK+RXERR,&URCTL ; Clear error flags
       RETI
;
; Interrupt handler for the UART Transmit part.
;
TXINT  TST.B  TXCNT                       ; Something to transmit?
       JZ     TXRET                       ; No, buffer is empty
       DEC.B  TXCNT                       ; Byte count –1
       PUSH   R5
       MOV    TXPOI,R5                    ; Pointer to buffer
       MOV.B  @R5+,&UTXBUF                ; Next byte for output
       MOV    R5,TXPOI                    ; Update pointer
       POP    R5
TXRET  RETI
```

```
;
; Interrupt Vectors
;
       .sect  ”SCIVEC”,0FFECh            ; USART interrupt vectors
       .word  TXINT                      ; Transmit vector
       .word  RCINT                      ; Receive vector
       .sect  ”INITVEC”,0FFFEh           ; Reset vector
       .word  INIT                       ; Program start address
```

3.2.2 ACLK Used as UART Clock

The following example deals with the ACLK used to generate the UART clock, or external frequencies lower than 100 kHz. Basically the same method used in section 3.2.1 can be applied to the ACLK used as the UART clock. See section 3.2.1 for an explanation.

This section shows a different approach: here the CPU is normally off, and leaves the LPM3 only when the number of received or transmitted characters programmed is reached.

EXAMPLE: full-duplex UART software code using the UART interrupt is shown. It is designed for:

- Baud rate: 2400 baud
- The ACLK (32,768Hz) used as the UART clock
- Eight data bits
- Two stop bits
- Parity enabled with odd-parity
- Receive of error-free characters only
- The CPU normally uses the low-power mode 3 (LPM3)

Transmit Part: the start address xxxx of the output sequence is loaded into pointer TXPOI, and the number of characters m is loaded into character count TXCNT. The interrupt routine outputs the character sequence and then, when TXCNT reaches zero (output completed), it resets the CPUoff bit of the stored status register on the stack. This manipulation omits the return to LPM3 and allows initialization of the next transmit sequence. R6 is exclusively used for the transmit part.

Receive Part: the start address yyyy of a RAM buffer is loaded into pointer RCPOI, and the number of characters n is loaded into character count RCCNT. The interrupt routine receives the characters and stores them in the buffer until RCCNT reaches zero (input completed). Then it resets the CPUoff bit of the stored status register on the stack. This manipulation omits the return to LPM3 and allows to process the received data. Only error-free characters are accepted. R7 is exclusively used for the receive part.

```
STACK     .equ   0600h                  ; Stack start address
;
; Definitions for the UART part
;
Baudr     .equ   2400                   ; Baud rate is 2400 Baud
FLLMPY    .equ   64                     ; FLL multiplier for 2,096MHz
UARTCLK   .equ   32768                  ; ACLK is used for UARTCLK
;
          .bss   TXCNT,1                ; Counter/status for transmit
          .bss   RCCNT,1                ; Counter/status for receive
          CALC_UMCTL                    ; Calculate Mod. Reg. content
          CALC_UBR                      ; Calculate UBR1/UBR0 contents
;
          .text                            ; Software start address
;
INIT      MOV    #STACK,SP              ; Initialize Stack Pointer
          CALL   #INITSR                ; Init. FLL and RAM
          ...                           ; Proceed with initialization
;
; Initialize the UART: Odd parity, 8 data bits, 2 stop bits
; ACLK used for the UART clock
;
          MOV.B  #CUMCTL,&UMCTL         ; Modulation Register
          MOV.B  #CUBR0,&UBR0           ; Baud rate register low
          MOV.B  #CUBR1,&UBR1           ; Baud rate register high
          BIS.B  #URXD+UTXD,&P4SEL      ; Select RXD + TXD at Port4
          BIS.B  #UTXE+URXE,&ME2        ; Enable USART modules
          MOV.B  #PENA+SP_+CHAR,&UCTL   ; USART control register
          MOV.B  #SSEL0,&UTCTL          ; Transmit contr. reg. ACLK
          MOV.B  #0,&URCTL              ; Receive control register
          BIS.B  #UTXIE+URXIE,&IE2      ; Enable USART interrupts
          CLR.B  TXCNT                  ; Disable transmit
          CLR.B  RCCNT                  ; Disable receive
          ...                           ; Continue with initialization
          EINT                          ; Enable interrupt (GIE = 1)
;
MAINLOOP  ...                           ; Start Mainloop
;
; Preparation for the reception of m bytes. Buffer starts
; at address yyyy. R7 is a dedicated register for receive
;
          TST.B  RCCNT                  ; Ready?
          JNZ    L$1                    ; No, RCCNT > 0
          MOV    #yyyy,R7               ; Receive buffer start address
          MOV.B  #m,RCCNT               ; Number of bytes
L$1 ...
;
; Stop the reception of data. The actually received character
; is input completely
;
          CLR.B  RCCNT                  ; Status is zero
          ...
;
; Preparation for transmission of n bytes starting at
; address xxxx. R6 is a dedicated register for transmit.
; The check for the empty TX buffer is faster, but needs more
; ROM bytes.
;
          TST.B  TXCNT                  ; Ready for next characters?
          JNZ    L$2                    ; No, TXCNT > 0
          BIT.B  #UTXIFG,&IFG2          ; TX part also ready?
          JZ     L$2                    ; No, busy
;
          MOV.B  #n–1,TXCNT             ; Ready, init. byte count
          MOV    #xxxx+1,R6             ; Init. transmit buffer pointer
          MOV.B  xxxx,&UTXBUF           ; First info byte to TX buffer
L$2       ...                           ; Continue in background
;
; Stop transmission of data. The actually sent character
; is transmitted completely
;
          CLR.B  TXCNT                  ; Status is zero
          ...
;
; After completion of all tasks, the program enters LPM3
;
PLPM3     BIS    #CPUoff+GIE+SCG1+SCG0,SR  ; Enter LPM3
;
; An interrupt handler cleared the CPUoff bit on the stack.
; Checks are made to see if activity is needed:
: Receive:   receive input buffer full
; Transmit:  transmit buffer output completely
; ...        other interrupt handlers
;
          TST.B  RCCNT                  ; Receive completed?
          JZ     PROCRC                 ; Yes, process received data
          TST.B  TXCNT                  ; Transmit completed?
          JZ     NXTTX                  ; Yes, prepare next characters
          ...                           ; Other handlers?
          JMP    PLPM3                  ; Back to LPM3
;
; Interrupt Handlers
; Interrupt handler for the UART Receive part. R7 is used
; only for the receive part.
;
RCINT     TST.B  RCCNT                  ; Reception allowed?
          JZ     RCRET                  ; No, status is 0
          BIT.B  #RXERR,&URCTL          ; Error during receive?
          JNZ    RCERR                  ; Yes
          DEC.B  RCCNT                  ; Byte count –1
          MOV.B  &URXBUF,0(R7)          ; Next byte to buffer
          INC    R7                     ; To next buffer byte
          TST.B  RCCNT                  ; Buffer filled?
          JNZ    RCRET                  ; No, proceed
          BIC    #CPUoff+SCI1+SCI0,0(SP); Active Mode after RETI
RCRET     RETI
;
RCERR     ...                           ; Error handling
          BIC.B  #FE+PE+OE+BRK+RXERR,&URCTL; Clear error flags
          RETI
;
; Interrupt handler for the UART Transmit part. R6 is used
; only for the transmit part
;
TXINT     TST.B  TXCNT                  ; Something to transmit?
          JZ     TXRET                  ; No, buffer is empty
          DEC.B  TXCNT                  ; Byte count –1
          MOV.B  @R6+,&UTXBUF           ; Next byte for output
          TST.B  TXCNT                  ; Buffer output?
          JNZ TXRET                     ; No, proceed
          BIC    #CPUoff+SCI1+SCI0,0(SP); Active Mode after RETI
TXRET  RETI
;
; Interrupt vectors
;
          sect   ”SCIVEC”,0FFECh        ; USART Interrupt vectors
          word   TXINT                  ; Transmit vector
          word   RCINT                  ; Receive vector
          sect   ”INITVEC”,0FFFEh       ; Reset vector
          word   INIT                   ; Program start address
```

3.3 Subroutines and .MACROs

This section contains the subroutines and assembler .MACROs used in the previous examples.

3.3.1 Subroutines

The initialization subroutine INITSR—which is explained in detail in the *MSP430 Application Report*—first checks if a power-up clear (PUC), or a power-on reset (POR) event has taken place:

- Power-Up Clear: the supply voltage is switched on, the RAM is cleared.
- Power-On Reset: a reset occurred (RST/NMI-pin or by watchdog), the RAM is not changed.

These two situations differ by the content of the INITKEY word. If it contains 0F05Ah, the power-on reset state is assumed; otherwise, the power-up clear state is assumed.

The subroutine selects the correct current switch FN_x for the system clock generator, and waits 30,000 clock cycles to ensure that it has locked to the correct oscillator tap.

```
; Common Initialization Subroutine
; Check the INITKEY value first:
; If value is 0F05Ah: a reset occurred, RAM is not cleared
; otherwise Vcc was switched on: complete initialization
;
INITSR    CMP    #0F05Ah,INITKEY        ; PUC or POR?
          JEQ    IN0                    ; Key is ok, continue program
          CALL   #RAMCLR                ; Restart completely: clear RAM
          MOV    #0F05Ah,INITKEY        ; Define “initialized state”
;
IN0       MOV.B  #FLLMPY–1,&SCFQCTL     ; Define MCLK frequency
;
          .if    FLLMPY < 48            ; Use the right DCO current:
          MOV.B  #0,&SCFI0              ; MCLK < 1.5MHz: FN_x off
          .else
          .if    FLLMPY < 80            ; 1.5MHz < MCLK < 2.5MHz?
          MOV.B  #FN_2,&SCFI0           ; Yes, FN_2 on
          .else                         ;
          .if    FLLMPY < 112           ; 2.5MHz < MCLK < 3.5MHz?
          MOV.B  #FN_3,&SCFI0           ; Yes, FN_3 on
          .else
          MOV.B  #FN_4,&SCFI0           ; MCLK > 3.5MHz: FN_4 on
          .endif
          .endif
          .endif
;
```

```
             MOV    #10000,R5          ; Allow the FLL to settle
IN1          DEC    R5                 ; at the correct DCO tap
             JNZ    IN1                ; during 30000 cycles
             RET                       ; Return from initialization
;
; Subroutine for clearing of the RAM block
;
             .bss   INITKEY,2,0200h    ; 0F05Ah: initialized state
RAMSTRT      .equ   0200h              ; Start of RAM
RAMEND       .equ   05FEh              ; Highest RAM address (33x)
;
RAMCLR       CLR    R5                 ; Prepare index register
RCL          CLR    RAMSTRT(R5)        ; 1st RAM address
             INCD   R5                 ; Next word address
             CMP    #RAMEND–RAMSTRT+2,R5 ; RAM cleared?
             JLO    RCL                ; No, once more
             RET                       ; Yes, return
```

3.3.2 .MACROs

The two following software macros calculate the best-fit values of the UART baud-rate generator. They do not use ROM or RAM, and they define the three variables, CUBR1, CUBR0, and CUMCTL, used during initialization of UART registers UBR1, UBR0, and UMCTL.

```
             .mnolist                  ; Do not list macro calls
;
; The values for the modulation registers UBR1/UBR0 are
; calculated: CUBR1 and CUBR0 contain the truncated result
; of the UARTCLK/Baudr division.
;
CALC_UBR     .macro
CUBR1        .equ   UARTCLK/(Baudr*256)    ;  Baud Rate Reg. High
CUBR0        .equ   (UARTCLK/Baudr)–256*CUBR1 ; Baud Rate Reg. Low
             .endm
```

The calculation of the modulation register UMCTL content follows. Seven bits of resolution are used.

```
CALC_UMCTL   .macro
;
; Modulation Register content: the rounded fraction of
; CMOD = UARTCLK/Baudr     is calculated
; Binary format of CMOD: 0.xxxxxxx
; Then the 8 bits of UMCTL are built.
; Inputs: UARTCLK, Baudr               ; Frequencies [Hz]
; Output: CUMCTL                       ; 8–bit UMCTL register value
;
```

```
CMOD      .equ   ((((256*UARTCLK)/Baudr)–256*(UARTCLK/Baudr))+1)/2
M$00      .equ   CMOD+CMOD              ; Fraction x 2
          .if    M$00>127               ; Overflow to integer?
M$10      .equ   M$00–128+CMOD          ; Yes, subtract 1.000000
C$0       .equ   1                      ; UMCTL.0 = 1
          .else
M$10      .equ   M$00+CMOD              ; No, add fraction
C$0       .equ   0                      ; UMCTL.0 = 0
          .endif
          .if    M$10>127               ; Overflow to integer?
M$20      .equ   M$10–128+CMOD          ; Yes, subtract 1.000000
C$1       .equ   2                      ; UMCTL.1 = 1
          .else
M$20      .equ   M$10+CMOD              ; No, add fraction
C$1       .equ   0                      ; UMCTL.1 = 0
          .endif
          .if    M$20>127               ; Overflow to integer?
M$30      .equ   M$20–128+CMOD          ; Yes, subtract 1.000000
C$2       .equ   4                      ; UMCTL.2 = 1
          .else
M$30      .equ   M$20+CMOD              ; No, add fraction
C$2       .equ   0                      ; UMCTL.2 = 0
          .endif
          .if    M$30>127               ; Overflow to integer?
M$40      .equ   M$30–128+CMOD          ; Yes, subtract 1.000000
C$3       .equ   8                      ; UMCTL.3 = 1
          .else
M$40      .equ   M$30+CMOD              ; No, add fraction
C$3       .equ   0                      ; UMCTL.3 = 0
          .endif
          .if    M$40>127               ; Overflow to integer?
M$50      .equ   M$40–128+CMOD          ; Yes, subtract 1.000000
C$4       .equ   10h                    ; UMCTL.4 = 1
          .else
M$50      .equ   M$40+CMOD              ; No, add fraction
C$4       .equ   0                      ; UMCTL.4 = 0
          .endif
          .if    M$50>127               ; Overflow to integer?
M$60      .equ   M$50–128+CMOD          ; Yes, subtract 1.000000
C$5       .equ   20h                    ; UMCTL.5 = 1
          .else
M$60      .equ   M$50+CMOD              ; No, add fraction
C$5       .equ   0                      ; UMCTL.5 = 0
```

```
          .endif
          .if    M$60>127               ; Overflow to integer?
M$70      .equ   M$60–128+CMOD          ; Yes, subtract 1.000000
C$6       .equ   40h                    ; UMCTL.6 = 1
          .else
M$70      .equ   M$60+CMOD              ; No, add fraction
C$6       .equ   0                      ; UMCTL.6 = 0
          .endif
          .if    M$70>127               ; Overflow to integer?
C$7       .equ   80h                    ; UMCTL.7 = 1
          .else
C$7       .equ   0                      ; UMCTL.7 = 0
          .endif
CUMCTL    .equ   C$7+C$6+C$5+C$4+C$3+C$2+C$1+C$0 ; Add bits
          .endm
```

4 References

1. *MSP430 Family Architecture Guide and Module Library*, 1996, Literature No. SLAUE10B
2. *MSP430 Application Report*, 1998, Literature No. SLAAE10C
3. Data Sheet MSP430x33x, 1998, Literature No. SLAS163

Appendix A Definitions

The abbreviations used in the hardware definitions are in conformance with the *Architecture User's Guide*, except for the stop bit definition (SP), which is a predefined symbol of the MSP430 assembler for the system Stack pointer (SP).

```
; HARDWARE DEFINITIONS
;
UCTL      .equ  070h  ; USART Control Register
SWRST     .equ  001h  ; 1: Software Reset 0: Run
SYNC      .equ  004h  ; 1: UART Mode      0: SCI Mode
CHAR      .equ  010h  ; 1: 8 Data Bits    0: 7 Data Bits
SP        .equ  020h  ; 1: 2 Stop Bits    0: 1 Stop Bit
PEV       .equ  040h  ; 1: Even Parity    0: Odd Parity
PENA      .equ  080h  ; 1: Parity enabled 0: Parity dis.
;
UTCTL     .equ  071h  ; Transmit Control Register
TXEPT     .equ  001h  ; 1: Transmitter empty
URXSE     .equ  008h  ;
SSEL0     .equ  010h  ; Clock Selection   0: Ext. Clock
SSEL1     .equ  020h  ; 1: ACLK           2,3: MCLK
;
URCTL     .equ  072h  ; Receive Control Register
RXERR     .equ  001h  ; 1: Receive Error  0: No Error
URXEIE    .equ  008h  ; 1: all Char.      0: only w/o Error
BRK       .equ  010h  ; 1: Break detected 0: ok
OE        .equ  020h  ; 1: Overrun Error  0: ok
PE        .equ  040h  ; 1: Parity Error   0: ok
FE        .equ  080h  ; 1: Frame Error    0: ok
;
UMCTL     .equ  073h  ; Modulation Control Reg. m7..m0
UBR0      .equ  074h  ; Baud Rate Register 0
UBR1      .equ  075h  ; Baud Rate Register 1
URXBUF    .equ  076h  ; Receive Buffer
UTXBUF    .equ  077h  ; Transmit Buffer
;
IFG2      .equ  003h  ; SFRs: Flags
URXIFG    .equ  001h  ; Receive Flag IFG2.0
UTXIFG    .equ  002h  ; Transmit Flag IFG2.1
;
IE2       .equ  001h  ; SFRs: Interrupt Enable Bits
URXIE     .equ  001h  ; Receive Intrpt Enable Bit IE2.0
UTXIE     .equ  002h  ; Transmit Intrpt Enable Bit IE2.1
;
ME2       .equ  005h  ; SFRs: Mode Enable Bits
```

```
URXE         .equ   001h   ; Receiver Module Enable Bit ME2.0
UTXE         .equ   002h   ; Transmitter Mod. Enable Bit ME2.1
;
P4SEL        .equ   01Fh   ; Port4 Sel. Reg. (I/O <–> USART)
URXD         .equ   080h   ; Receive Input P4.7
UTXD         .equ   040h   ; Transmit Output P4.6
;
SCG1         .equ   080h   ; Low Power Mode bit 1
SCG0         .equ   040h   ; Low Power Mode bit 0
CPUoff       .equ   010h   ; Switches CPU off
GIE          .equ   008h   ; General Interrupt Enable Bit
;
SCFQCTL      .equ   052h   ; FLL multiplier and M bit
SCFI0        .equ   050h   ; FLL current switches
FN_2         .equ   004h   ; Switch for 2 MHz
FN_3         .equ   008h   ; Switch for 3 MHz
FN_4         .equ   010h   ; Switch for 4 MHz
```