# Dual UART Virtual Peripheral Implementation

**Application Note 39**

November 2000

## 1.0 Introduction

The Dual UART Virtual Peripheral uses the SX communications controller to provide asynchronous data communication through the RS-232 interface. This Virtual Peripheral makes the SX device act as a Universal Asynchronous Transmitter and Receiver. The Virtual Peripheral has been developed using the SX Evaluation Board and has been tested using the SX-Key of Parallax Inc. and SX-IDE of Advanced Transdata Inc.

Unlike other MCUs that add functions in the form of additional silicon, the SX Series uses its industry-leading performance to execute functions as software module, or Virtual Peripheral. These are loaded into a high-speed on-chip flash/EEPROM program memory and executed as required. In addition, a set of on-chip hardware Peripherals is available to perform operations that cannot readily be done in software, such as comparators, timers and oscillators.

## 2.0 Description of Dual UART Virtual Peripheral

The data transmission is done at a pre determined baud rate. This is done by over sampling the data to be transmitted. A divide ratio is calculated by dividing this sampling rate by the required baud rate. The data is then inverted before it is sent at RS-232 levels through the line driver. The two UART Virtual peripheral modules work simultaneously at different baud rates. As data has to be sent on both the UART's simultaneously at different Baud rates it is necessary that the user checks that the transmit flag of the particular UART is reset before he sends any data on it, so that data corruption by overwriting of data is prevented. As the Virtual Peripheral is configured to send data on both the UART's simultaneously, alot of time is saved when compared to the sequential type of operation. At the occurrence of every interrupt the Virtual Peripheral checks for any data that is to be received on both the UART's and if it receives data it waits till it receives a complete byte of data(8 bits) and then sets a Flag indicating that a byte of data has been received on the particular UART.

### 2.1 Program Description

Multithreading concept is used in this Virtual Peripheral to realize the UART. Whenever an RTCC interrupt occurs the program jumps into the interrupt routine, which contains the interrupt multitasker. The multitasker has a number of threads (normally 24 or less). In the current implementation for the UART Virtual Peripheral , we are using 16 threads. At every occurrence of the interrupt, the interrupt control jumps to one of the threads. The virtual peripherals are inserted into one of the thread. In the UART Virtual Peripheral, it is included in the isrThread1. As this thread is called once in 4 times, the UART routines executes once for every 4 interrupts. This technique enables the user to include other Virtual Peripheral modules in the remaining threads. Before sending any byte the user must take care to check whether the transmit flag is cleared and then he must set the transmit flag before he calls the "sendbyte" routine. The Virtual peripheral also features the capability to send strings that are stored in the area allocated for strings.

Note:Both the UART Virtual Peripheral modules are in isrThread1. The program can be modified to include one UART in isrThread1 & the other in isrThread2. The ISR jump table have to be modified to include isrThread2 very 4th cycle of the interrupt, as is done for isrThread1.

## 2.2 Interrupt Service Routine Flowchart

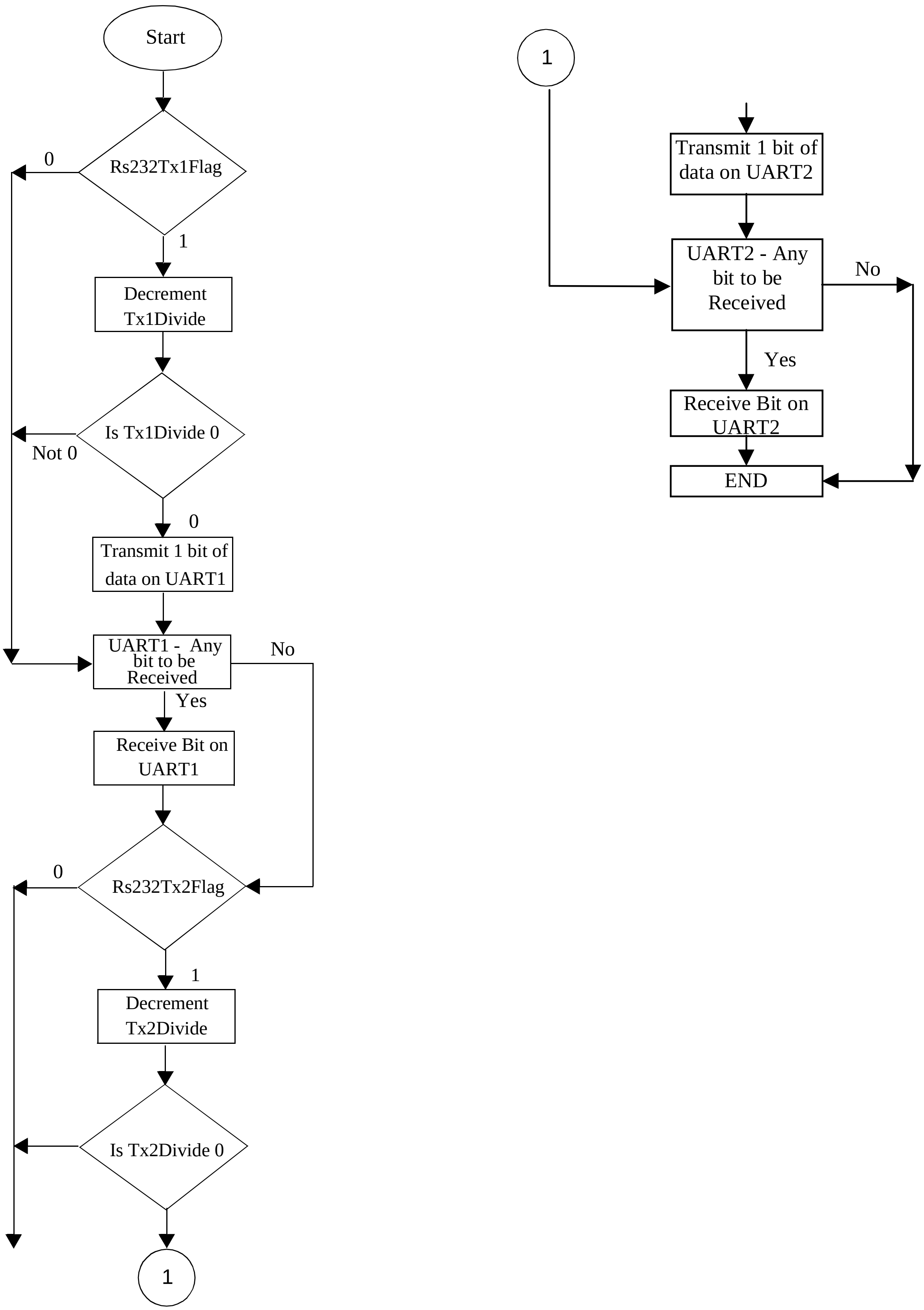


**Figure 2-1. Interrupt Service Routine Flowchart**

## 3.0 Different Sections of UART Virtual Peripheral

This documentation provides a brief overview of different sections involved in "Dual UART Virtual Peripheral Using SX Communications Controller".

The four sections of the Dual UART Virtual Peripheral module mentioned below can be inserted in a main source code at appropriate locations to meet the requirement of realization of the UART Virtual Peripheral.

- Equates Section
- Bank Section
- Initialization Section
- Interrupt Section

### 3.1 Equates Section

This section gives the equates section of the Dual UART Virtual Peripheral module and it also defines the output pins for the Dual UART Virtual Peripheral. The value of UARTDivide, UARTStDelay and pin declarations are made here.

The value of the constants are as follows:

UARTfs = 230400

Num = 4

Int Period = 217

UARTDivide1 = UARTfs/(UARTbaud1 * Num)

UARTStDelay1 = UARTDivide1 + (UARTDivide1/2)+1

UARTDivide2 = UARTfs/(UARTbaud2 * Num)

UARTStDelay2 = UARTDivide2 + (UARTDivide2/2)+1

Where Num is the number of times the ISR thread in which the Virtual Peripheral is present is called in the Interrupt service routine multitasker (ISR multiplexer which is 4 in our case).

The pins on which the input and output data are received and sent are defined in this section. Port Ra and Rc are used for the external interface.

The Pins are configured as follows:

```
rs232Rxpin1  equ   ra.2  ;UART1 receive input
rs232Txpin1  equ   ra.3  ;UART1 transmit output
rs232Rxpin2  equ   rc.7  ;UART2 receive input
rs232Txpin2  equ   rc.6  ;UART2 transmit output
```

The baud rates for each of the UART's are decided by using the IFDEF statements, depending on the baud rate selected. The Baud rate is equal to the number that represents it in the commented statement. For example, if U1B1200 is uncommented it implies that UART1 has a baud rate of 1200bps, similarly U2B1920 implies UART2 is to be configured for a baud rate of 19200bps.

### 3.2 Bank Section

This section describes the use of the banks in the UART Virtual Peripheral. Two banks that are used in the UART Virtual Peripheral module (bank1, bank2).

Inside bank1 we have defined different banks for `rs232TxBank` and `MultiplexBank` just for clarity, and bank2 contains `rs232RxBank`.

```
       Org    bank1_org


bank1               =       $
rs232TxBank         =       $     ;UART Transmit bank
rs232Tx1high        ds      1     ;High Byte to be transmitted
rs232Tx1low         ds      1     ;Low Byte to be transmitted
rs232Tx1count       ds      1     ;counter to keep track of the bits sent
rs232Tx1divide      ds      1     ;xmit timing counter
rs232Tx1Byte        ds      1     ;store the byte to be sent in this buffer

rs232Tx2high        ds      1     ;High Byte to be transmitted
rs232Tx2low         ds      1     ;Low Byte to be transmitted
rs232Tx2count       ds      1     ;counter to keep track of the bits sent
rs232Tx2divide      ds      1     ;xmit timing counter
rs232Tx2Byte        ds      1     ;store the byte to be sent in this buffer

MultiplexBank       =      $      ;Multipler Bank
isrMultiplex        ds     1      ;Used to jump between the Isr Threads when
                                  ; An Interrupt occurs

       Org    bank2_org

bank2               =      $
rs232RxBank         =      $
rs232Rx1count       ds      1     ;counter to keep track of the number of bits received
rs232Rx1divide      ds      1     ;receive timing counter
rs232Rx1byte        ds      1     ;buffer for incoming byte
rs232byte1          ds      1     ;Used by serial routines
rs232Rx2count       ds      1     ;counter to keep track of the number of bits received
rs232Rx2divide      ds      1     ;receive timing counter
rs232Rx2byte        ds      1     ;buffer for incoming byte
rs232byte2          ds      1     ;used by serial routines
```

### 3.3 Initialization Section

This provides the initialization part of the Dual UART Virtual Peripheral . This has to be included before the main loop starts with the initialization of all other ports and registers.

```
_bank rs232TxBank                 ;select rs232 bank

mov w,#UARTdivide1                ;load Txdivide with UART baud rate
mov rs232Txdivide1,w

mov w,#UARTdivide2                ;load Txdivide with UART baud rate
mov rs232Txdivide2,w
```

This initialization is done to send the data at the required baud rate. The value of UARTDivide symbolizes the number of times the interrupt has to be serviced before a bit is transmitted. For example if we are transmitting data at a rate of 9600bps, the value of UARTDivide is 6, this means that every one bit should be transmitted for every 6 occurrences of isrThread1.

### 3.4 Interrupt Section

The flow of the interrupt service routine is shown in Figure 2-1.

The interrupt service routine of the UART Virtual Peripheral module with a "retiw" value of -217 at an oscillator frequency of 50MHz runs every 4.32us.

```
;*****************************************************************************************
             org    INTERRUPT_ORG              ; First location in program memory.
;*****************************************************************************************
;*****************************************************************************************
;------------------------------------ Interrupt Service Routine -------------------------------
; Note: The interrupt code must always originate at address $0.
; Interrupt Frequency = (Cycle Frequency / -(retiw value))  For example:
; With a retiw value of -217 and an oscillator frequency of 50MHz, this
; code runs every 4.32us.
;*****************************************************************************************
             org     $0
interrupt                                      ;3

;*****************************************************************************************
; Interrupt
; Interrupt Frequency = (Cycle Frequency / -(retiw value))  For example:
; With a retiw value of -217 and an oscillator frequency of 50MHz, this code runs
; every 4.32us.
;*****************************************************************************************

;*****************************************************************************************
;---------------------------------------------VP:VP Multitasker--------------------------------
; Virtual Peripheral Multitasker : up to 16 individual threads, each running at the
; (interrupt rate/16). Change then below:
;Input variable(s): isrMultiplex: variable used to choose threads
;Output variable(s): None,executes the next thread
;Variable(s) affected: isrMultiplex
;Flag(s) affected: None
;Program Cycles: 9 cycles (turbo mode)
;*****************************************************************************************
                _bank          Multiplexbank        ;
                inc            isrMultiplex         ; toggle interrupt rate
                mov            w,isrMultiplex       ;
;*****************************************************************************************
; The code between the tableStart and tableEnd statements MUST be completely within the first
; half of a page. The routines it is jumping to must be in the same page as this table.
;*****************************************************************************************
         tableStart                                 ; Start all tables with this macro
                jmp            pc+w                 ;
                jmp            isrThread1           ;
                jmp            isrThread2           ;
                jmp            isrThread3           ;
                jmp            isrThread4           ;
                jmp            isrThread1           ;
                jmp            isrThread5           ;
                jmp            isrThread6           ;
                jmp            isrThread7           ;
                jmp            isrThread1           ;
                jmp            isrThread8           ;
                jmp            isrThread9           ;
                jmp            isrThread10          ;
                jmp            isrThread1           ;
```

```
                jmp         isrThread11         ;
                jmp         isrThread12         ;
                jmp         isrThread13         ;
         tableEnd                               ; End all tables with this macro.
;**************************************************************************************************
;VP: VP Multitasker
; ISR TASKS
;**************************************************************************************************
isrThread1                                      ; Serviced at ISR rate/4
;**************************************************************************************************
; Virtual Peripheral: Universal Asynchronous Receiver Transmitter (UART) These routines send
; and receive RS232 serial data, and are currently configured (though modifications can be
; made) for the popular "No parity-checking, 8 data bit, 1 stop bit" (N,8,1) data format.
;
; The VP below has 2 UARTS implemented - UART1 & UART2. Both the UARTs can work at independent
; Baud Rates.
;
; RECEIVING: The rs232Rx1flag & rs232Rx2flag are set high whenever a valid byte of data has
; been received and it is the calling routine's responsibility to reset this flag once the
; incoming data has been collected.
;
; TRANSMITTING: The transmit routine requires the data to be inverted and loaded
; (rs232Txhigh+rs232Txlow) register pair (with the inverted 8 data bits stored in
; rs232Txhigh and rs232Txlow bit 7 set high to act as a start bit). Then the number of bits
; ready for transmission (10=1 start + 8 data + 1 stop) must be loaded into the rs232Txcount
; register. As soon as this latter is done, the transmit routine immediately begins sending
; the data. This routine has a varying execution rate and therefore should always be
; placed after any timing-critical virtual peripherals such as timers,
; adcs, pwms, etc.
; Note:The transmit and receive routines are independent and either may be removed for each
;      of the UARTs. The initial "_bank rs232TxBank" & "_bank rs232RxBank" (common)
;      instruction is kept for Transmit & Receive routines.
;       Input variable(s):       rs232Tx1Low (only high bit used), rs232Tx1High, rs232Tx1Count
;                                If rs232Tx1Flag SET, then transmit on UART1
;                                rs232Tx2Low (only high bit used), rs232Tx2High, rs232Tx2Count
;                                If rs232Tx2Flag SET, then transmit on UART2
;       Output variable(s):      rs232Rx1Flag, rs232Rx1byte
;                                rs232Rx2Flag, rs232Rx2byte
;       Variable(s) affected:    rs232Tx1divide, rs232Rx1divide, rs232Rx1count
;                                rs232Tx2divide, rs232Rx2divide, rs232Rx2count
;       Flag(s) affected:        rs232Tx1Flag, rs232Tx2Flag
;                                rs232Rx1Flag, rs232Rx1Flag
;      Program cycles:           22 worst case for Tx, 23 worst case for Rx
;      Variable Length?          Yes.
;**************************************************************************************************
UART1
rs232Transmit
             _bank        rs232TxBank         ;2 switch to serial register bank

             sb           rs232Tx1FLag        ;1
             jmp          rs232Receive1       ;1
             decsz        rs232Tx1divide      ;1 only execute the transmit routine
             jmp          rs232Receive1       ;1
             mov          w,#UARTDivide1      ;1 load UART baud rate (50MHz)
             mov          rs232Tx1divide,w    ;1
             test         rs232Tx1count       ;1 are we sending?
             snz                              ;1
```

```
            jmp         rs232Receive1       ;1

:txbit      clc                             ;1 yes, ready stop bit
            rr          rs232Tx1high        ;1 and shift to next bit
            rr          rs232Tx1low         ;1
            dec         rs232Tx1count       ;1 decrement bit counter
            snb         rs232Tx1low.6       ;1 output next bit
            clrb        rs232TxPin1         ;1
            sb          rs232Tx1low.6       ;1
            setb        rs232TxPin1         ;1
            test        rs232Tx1count       ;1 are we sending?
            snz                             ;1
            clrb        rs232Tx1Flag        ;1,22

rs232Receive1
            _bank       rs232RxBank         ;2
            sb          rs232RxPin1         ;1 get current rx bit
            clc                             ;1
            snb         rs232RxPin1         ;1
            stc                             ;1
            test        rs232Rx1count       ;1 currently receiving byte?
            sz                              ;1
            jmp         :rxbit              ;1 if so, jump ahead
            mov         w,#9                ;1 in case start, ready 9 bits
            sc                              ;1 skip ahead if not start bit
            mov         rs232Rx1count,w     ;1 it is, so renew bit count
            mov         w,#UARTStDelay1     ;1 ready 1.5 bit periods (50MHz)
            mov         rs232Rx1divide,w    ;1
:rxbit      decsz       rs232Rx1civide      ;1 middle of next bit?
            jmp         :rs232RxOut1        ;1
            mov         w,#UARTDivide1      ;1 yes, ready 1 bit period (50MHz)
            mov         rs232Rx1divide,w    ;1
            dec         rs232Rx1count       ;1 last bit?
            sz                              ;1 if not
            rr          rs232Rx1byte        ;1 then save bit
            snz                             ;1 if so,
            setb        rs232Rx1Flag        ;1,23 then set flag
:rs232RxOut1


UART2
            _bank       rs232TxBank         ;2 switch to serial register bank
            sb          rs232Tx2flag        ;1
            jmp         rs232Receive2       ;1
            decsz       rs232Tx2divide      ;1 only execute the transmit routine
            jmp         rs232Receive2       ;1
            mov         w,#UARTDivide2      ;1 load UART baud rate (50MHz)
            mov         rs232Tx2divide,w    ;1
            test        rs232Tx2count       ;1 are we sending?
            snz                             ;1
            jmp         rs232Receive2       ;1

:txbit      clc                             ;1 yes, ready stop bit
            rr          rs232Tx2high        ;1 and shift to next bit
            rr          rs232Tx2low         ;1
            dec         rs232Tx2count       ;1 decrement bit counter
            snb         rs232Tx2low.6       ;1 output next bit
```

```
             clrb         rs232TxPin2          ;1
             sb           rs232Tx2low.6        ;1
             setb         rs232TxPin2          ;1
             test         rs232Tx2count        ;1 are we sending?
             snz                               ;1
             clrb         rs232Tx2Flag         ;1,22

rs232Receive2
             _bank        rs232RxBank          ;2
             sb           rs232RxPin2          ;1 get current rx bit
             clc                               ;1
             snb          rs232RxPin2          ;1
             stc                               ;1
             test         rs232Rx2count        ;1 currently receiving byte?
             sz                                ;1
             jmp          :rxbit               ;1 if so, jump ahead
             mov          w,#9                 ;1 in case start, ready 9 bits
             sc                                ;1 skip ahead if not start bit
             mov          rs232Rx2count,w      ;1 it is, so renew bit count
             mov          w,#UARTStDelay2      ;1 ready 1.5 bit periods (50MHz)
             mov          rs232Rx2divide,w     ;1
:rxbit       decsz        rs232Rx2civide       ;1 middle of next bit?
             jmp          :rs232RxOut2         ;1
             mov          w,#UARTDivide2       ;1 yes, ready 1 bit period (50MHz)
             mov          rs232Rx2divide,w     ;1
             dec          rs232Rx2count        ;1 last bit?
             sz                                ;1 if not
             rr           rs232Rx2byte         ;1 then save bit
             snz                               ;1 if so,
             setb         rs232Rx2Flag         ;1,23 then set flag
:rs232RxOut2

UARTOut

;*********************************************************************************************
;================= PUT YOUR OWN VPs HERE=====================
; Virtual Peripheral:
;
;      Input variable(s):
;      Output variable(s):
;      Variable(s) affected:
;      Flag(s) affected:
;*********************************************************************************************
;---------------------------------------------------------------------------------------------
             jmp          isrOut       ; 7 cycles until mainline program resumes execution
;---------------------------------------------------------------------------------------------
isrThread2                             ; Serviced at ISR rate/16
;---------------------------------------------------------------------------------------------
             jmp          isrOut       ; 7 cycles until mainline program resumes execution
;---------------------------------------------------------------------------------------------
isrThread3                             ; Serviced at ISR rate/16
;---------------------------------------------------------------------------------------------
             jmp          isrOut       ; 7 cycles until mainline program resumes execution
;---------------------------------------------------------------------------------------------
isrThread4                             ; Serviced at ISR rate/16
;---------------------------------------------------------------------------------------------
             jmp          isrOut       ; 7 cycles until mainline program resumes execution
```

```
;-------------------------------------------------------------------------------------------
isrThread5                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------------------
             jmp          isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------------------
isrThread6                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------------------
             jmp          isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------------------
isrThread7                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------------------
             jmp          isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------------------
isrThread8                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------------------
             jmp          isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------------------
isrThread9                              ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------------------
             jmp          isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------------------
isrThread10                             ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------------------
             jmp          isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------------------
isrThread11                             ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------------------
             jmp          isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------------------
isrThread12                             ; Serviced at ISR rate/16
;-------------------------------------------------------------------------------------------
             jmp          isrOut        ; 7 cycles until mainline program resumes execution
;-------------------------------------------------------------------------------------------
isrThread13                             ; Serviced at ISR rate/16
                                        ; This thread must reload the isrMultiplex register
             _bank  Multiplexbank
             mov    isrMultiplex,#255   ;reload isrMultiplex so isrThread1 will be run on the
                                        ; next interrupt.
             jmp    isrOut              ; 7 cycles until mainline program resumes execution
                                        ; This thread must reload the isrMultiplex register
                                        ; since it is the last one to run in a rotation.
;-------------------------------------------------------------------------------------------
isrOut

;*********************************************************************************
; Set Interrupt Rate
;*********************************************************************************

isr_end
             mov    w,#-intperiod       ;refresh RTCC on return
                                        ;(RTCC = 217 no of instructions executed in the ISR)
             retiw                      ;return from the interrupt

;*****************************************************************************************
; End of the Interrupt Service Routine
;*****************************************************************************************
```

## 4.0 Features

### 4.1 Baud Rate Generation Methodology and Timing

To understand the method for generating the required baud rate let us take an example.

Let us consider data has to be transmitted at the rate of 57600bps and the sampling frequency is 230.4KHz

The time taken for the transmission of 1 bit of data is equal to (1/57600)sec.

Data is sampled at a frequency of 4 * 57,600bps = 230.4KHz.

If data is sent at the sample rate, then it will be transmitted at a rate much faster than that required and hence will result in a baud rate mismatch. To avoid this mismatch we introduce a delay factor that is a ratio of the sampling frequency and baud rate.

Hence the divide ratio UARTDivide for the above example will be = (230400/57600) = 4

This divide ratio implies that if a bit of data is transmitted once in 4 occurrences of the interrupt, the baud rate matching will be taken care of.

When the concept of ISR thread is used it is necessary that the value of UARTDivide is further divide by a value equal to the number of times the thread servicing this particular interrupt is called in the ISR Multitasker.

As in the interrupt routine Mentioned above if the thread 1 is being called 4 times in the Interrupt Multitasker then the value of UARTDivide is further divided by 4 to get a resulting value of 1.

So the formula for UARTDivide will be:

UARTDivide = UARTfs/(UARTBaudrate*number of times the ISR is called in the Multitasker)

This gives a value for UARTDivide of 1. Hence this value will take care of the transmission of data at the required baudrate.

While receiving data, the timing is controlled in the same way as explained above.

The only difference is that a constant called UARTStDelay is introduced which is equal to 1.5 times the bit length. Its purpose is to ensure that the bits are sampled near the middle of each pulse which will ensure that the data is sampled accurately. Separate UARTDivide and UARTStDelay constants are computed for feasibility of both UART's being independent of each other. (e.g.: UARTStDelay1 refers to UART1, and UARTStDelay2 refers to UART2)

### 4.2 Circuit Design Procedure

The simplest version of the circuit requires two Port lines of the SX for TX & RX (if handshake is to be used, additional port lines will be required). The circuit interface is quite simple which involves only a driver for driving the signals. As we intend to use the RS-232 level of communication, any TTL to RS232 converter can be used. The TX and RX lines are to be given to the driver directly which takes care of the level conversion.

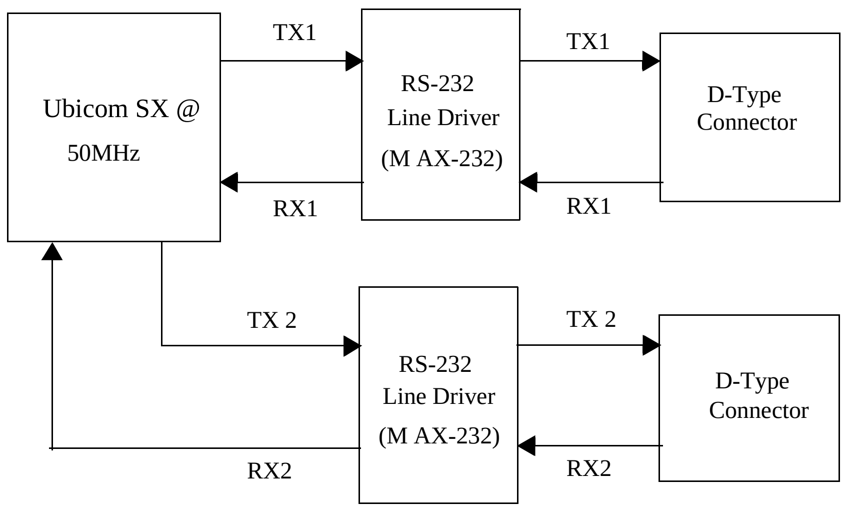


**Figure 4-1. Circuit Diagram**

## 5.0 Applications

The Applications of UART are innumerable and the use of UART is reflected in nearly every communication application. As UART is used for serial communication, we are using the most widely used serial communication standard that is the RS-232 standard.

The program written is for a simple UART without any handshakes. The program can be modified to handle the handshakes too.

As this implementation has 2 UART's which can be configured for different baud rates, it is possible to send messages quickly as both the UARTs are independent of each other. Hence it can be used in Applications where we have 2 MCU's or peripherals operating at different baud rates.The Virtual Peripheral can be modified by including the transmit and receive routines in different Isr threads so that the time taken to service thread1 will become less.

 

Lit #: AN39-01

## Sales and Tech Support Contact Information

For the latest contact and support information on SX devices, please visit the Ubicom website at www.ubicom.com. The site contains technical literature, local sales contacts, tech support and many other features.

**1330 Charleston Road**
**Mountain View, CA 94043**
Contact: Sales@ubicom.com
http://www.ubicom.com
Tel.: (650) 210-1500
Fax: (650) 210-8715