8-UART Virtual Peripheral Implementation

Application Note 40
April 2001

1.0 Introduction

The 8-UART Virtual peripheral uses the SX communications controller to provide asynchronous data communication for up to eight RS-232 interfaces. The Virtual Peripheral has been developed using the SX Evaluation Board and has been tested using the SX-Key interface from Parallax Inc. and the SXIDE integrated development environment from Advanced Transdata Inc.

Unlike other MCUs that add functions in the form of additional silicon, the SX Series uses its fast execution rate to emulate peripheral functions in software modules, called Virtual Peripherals. On-chip hardware peripherals are only provided for functions that cannot be performed efficiently in software, such as timers and analog comparators.

1.1 Program Description

The 8-UART Virtual Peripheral implements eight UART interfaces that can run at independent baud rates. Because all eight UARTs operate simultaneously, data transfer is much more efficient than implementations that only handle one channel at a time.

The 8-UART Virtual Peripheral is designed to operate in a multithreaded environment driven by the real-time clock/counter (RTCC). Whenever an RTCC interrupt occurs, an interrupt service routine (ISR) is called which contains a multitasker for allocating CPU bandwidth among any Virtual Peripherals which require interrupt service. Each task is called a *thread*, and the 8-UART Virtual Peripheral uses four threads that each handle two UART interfaces. The threads are called `isrThread1`, `isrThread2`, `isrThread3`, and `isrThread4`.

Before sending a character, software must check the transmit flag for the UART to be used. If the flag is clear, a character can be sent by setting the flag and calling the `sendbyte` routine. The Virtual Peripheral also features the capability to send strings.

In the ISR multitasker, there are only four threads. Other user Virtual Peripheral modules can be included within the present four threads or new threads can be added and the `Num` value changed accordingly.

1.2 Interrupt Service Routine Flowchart for Thread n with n = 1, 2, 3, 4

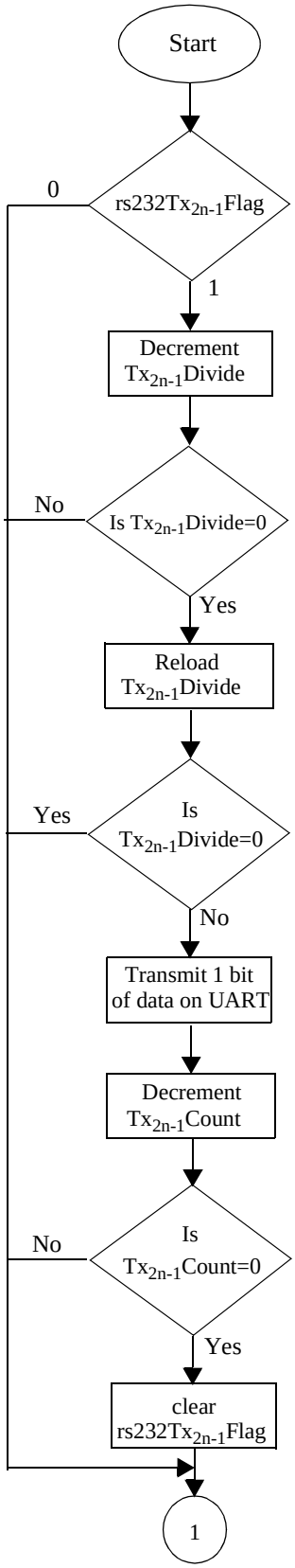

Figure 1. Interrupt Service Routine

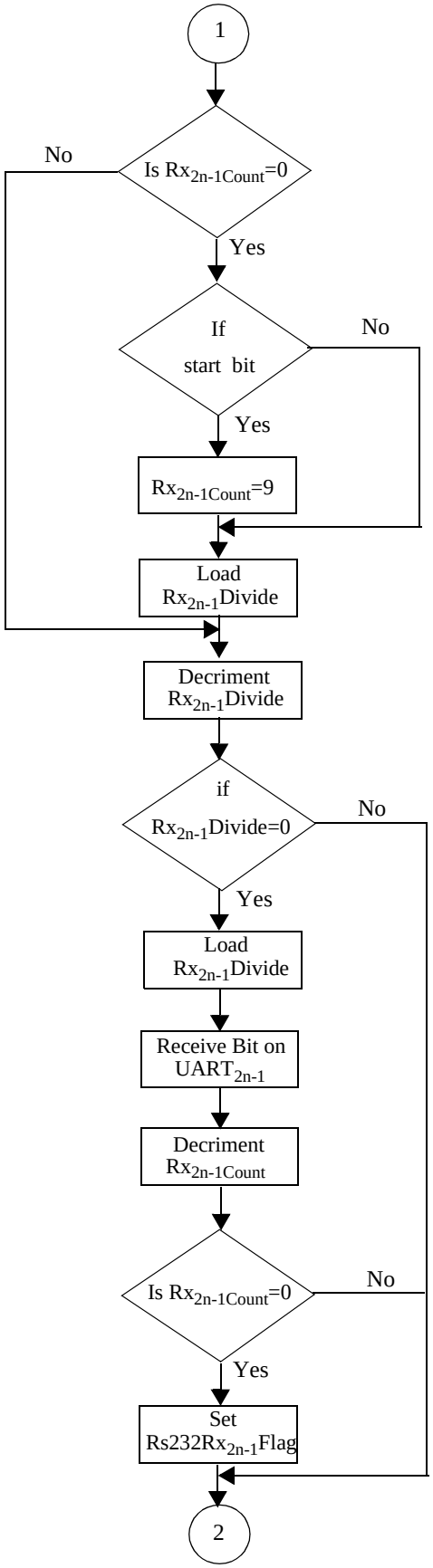

Figure 1-2. Interrupt Service Routine (continued)

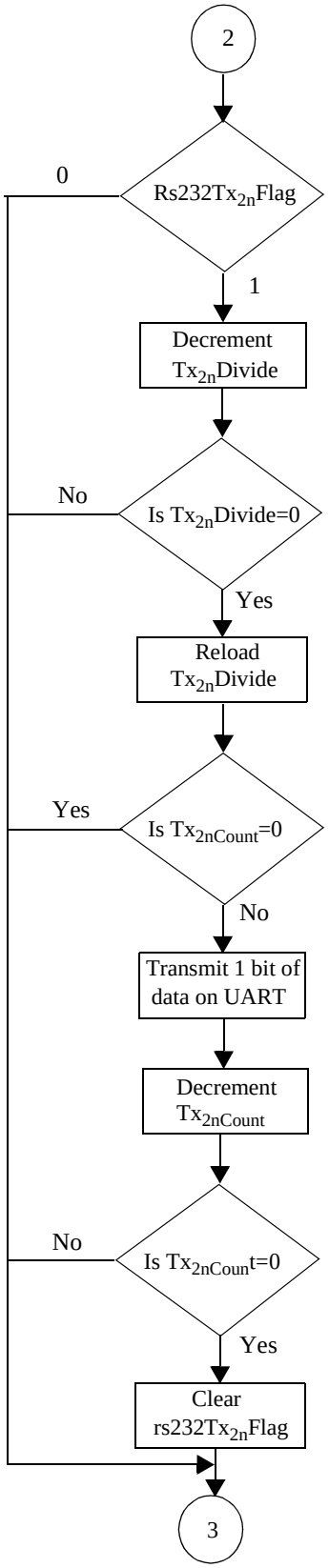

Figure 1-3. Interrupt Service Routine (continued)

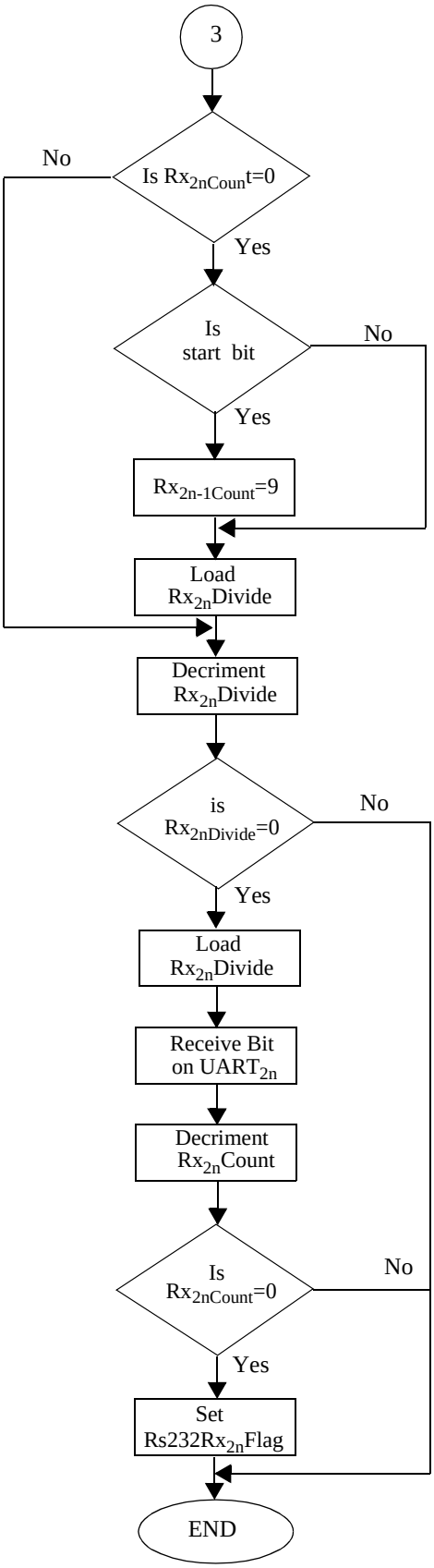

Figure 1-4. Interrupt Service Routine (continued)

2.0 Different Sections of UART Virtual Peripheral

The source code for the UART Virtual Peripheral is divided into four sections:

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

When integrated into an application, each section of the source code is inserted at an appropriate location in the main body of the application's source code.

2.1 Equates Section

The equates section provides the values of `UARTDivide` and `UARTStDelay` and the port pin declarations.

The values of the constants are as follows:

```
UARTfs        =      230400
Num           =      4
Int Period    =      217
UARTDividen = UARTfs/(UARTBaudn * Num)
UARTStDelayn = UARTDividen + (UARTDividen/2)+1
```

n = 1, 2, 3, 4, 5, 6, 7, or 8

`Num` is the number of times the UART Virtual Peripheral ISR is called by the multitasker during one rotation. The multitasker rotates interrupt service among four slots, and the 8-UART Virtual Peripheral is called from all four of these slots, so `Num` is 4 in this example. In other applications, `Num` might have a different value. For example, if the interrupt frequency were faster or the baud rate were slower, one slot might be sufficient to service the 8-UART Virtual Peripheral ISR.

The pins for sending and receiving data are defined in this section. Port A, Port B, and Port C 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       rb.2      ;UART2 receive input
rs232Txpin2     equ       rb.3      ;UART2 transmit output
rs232Rxpin3     equ       rb.4      ;UART3 receive input
rs232Txpin3     equ       rb.5      ;UART3 transmit output
rs232Rxpin4     equ       rb.6      ;UART4 receive input
rs232Txpin4     equ       rb.7      ;UART4 transmit output
rs232Rxpin5     equ       rc.0      ;UART5 receive input
rs232Txpin5     equ       rc.1      ;UART5 transmit output
rs232Rxpin6     equ       rc.2      ;UART6 receive input
rs232Txpin6     equ       rc.3      ;UART6 transmit output
rs232Rxpin7     equ       rc.4      ;UART7 receive input
rs232Txpin7     equ       rc.5      ;UART7 transmit output
rs232Rxpin8     equ       rc.6      ;UART8 receive input
rs232Txpin8     equ       rc.7      ;UART8 transmit output
```

The baud rates for each of the UARTs are specified by using IFDEF statements. The baud rate is equal to the number that represents it in the commented statement.

For example, if the `uart1baud1920` is uncommented, UART 1 is configured for a baud rate of 19200 baud. Similarly, if `uart2baud9600` is uncommented, UART 2 is configured for a baud rate of 9600 baud.

2.2 Bank Section

This section describes the use of the banks in the 8 UART Virtual Peripheral implementation. 5 banks are used in the 8 UART Virtual Peripheral module (BANK1 to BANK5). BANK1 and BANK2 are used for defining all the variables of the 8 transmit routines of the UART and BANK3 and BANK4 are used for defining all the variables of the 8 receive routines of the UART.

All the flags are defined in the global register bank.

```
     org     global_org
;----------------------------VP: RS232 Transmit ----------------------------------------
     flags0          equ        global_org + 0
     rs232Tx1Flag    equ        flags0.0        ;indicates the Uart1 tx
     rs232Tx2Flag    equ        flags0.1        ;indicates the Uart2 tx
     rs232Tx3Flag    equ        flags0.2        ;indicates the Uart3 tx
     rs232Tx4Flag    equ        flags0.3        ;indicates the Uart4 tx
     rs232Tx5Flag    equ        flags0.4        ;indicates the Uart5 tx
     rs232Tx6Flag    equ        flags0.5        ;indicates the Uart6 tx
     rs232Tx7Flag    equ        flags0.6        ;indicates the Uart7 tx
     rs232Tx8Flag    equ        flags0.7        ;indicates the Uart8 tx
;------------------------------- VP: RS232 Receive -----------------------------------
     flags1          equ        global_org + 1
     rs232RxFlag1    equ        flags1.0        ;indicates the reception of a bit from the UART1
     rs232RxFlag2    equ        flags1.1        ;indicates the reception of a bit from the UART2
     rs232RxFlag3    equ        flags1.2        ;indicates the reception of a bit from the UART3
     rs232RxFlag4    equ        flags1.3        ;indicates the reception of a bit from the UART4
     rs232RxFlag5    equ        flags1.4        ;indicates the reception of a bit from the UART5
     rs232RxFlag6    equ        flags1.5        ;indicates the reception of a bit from the UART6
     rs232RxFlag7    equ        flags1.6        ;indicates the reception of a bit from the UART7
     rs232RxFlag8    equ        flags1.7        ;indicates the reception of a bit from the UART8


     org   bank1_org

     bank1                =          $
     rs232TxBank1234      =          $          ;UART bank
     rs232Txhigh1         ds         1          ;hi byte to transmit
     rs232Txlow1          ds         1          ;low byte to transmit
     rs232Txcount1        ds         1          ;number of bits sent
     rs232Txdivide1       ds         1          ;xmit timing (/16) counter
     rs232Txhigh2         ds         1          ;hi byte to transmit
     rs232Txlow2          ds         1          ;low byte to transmit
     rs232Txcount2        ds         1          ;number of bits sent
     rs232Txdivide2       ds         1          ;xmit timing (/16) counter
     rs232Txhigh3         ds         1          ;hi byte to transmit
     rs232Txlow3          ds         1          ;low byte to transmit
     rs232Txcount3        ds         1          ;number of bits sent
     rs232Txdivide3       ds         1          ;xmit timing (/16) counter
     rs232Txhigh4         ds         1          ;hi byte to transmit
     rs232Txlow4          ds         1          ;low byte to transmit
     rs232Txcount4        ds         1          ;number of bits sent
     rs232Txdivide4       ds         1          ;xmit timing (/16) counter


     org   bank2_org

     bank2                =          $
     rs232TxBank5678      =          $          ;UART bank
     rs232Txhigh5         ds         1          ;hi byte to transmit
```

```
rs232Txlow5          ds        1          ;low byte to transmit
rs232Txcount5        ds        1          ;number of bits sent
rs232Txdivide5       ds        1          ;xmit timing (/16) counter
rs232Txhigh6         ds        1          ;hi byte to transmit
rs232Txlow6          ds        1          ;low byte to transmit
rs232Txcount6        ds        1          ;number of bits sent
rs232Txdivide6       ds        1          ;xmit timing (/16) counter
rs232Txhigh7         ds        1          ;hi byte to transmit
rs232Txlow7          ds        1          ;low byte to transmit
rs232Txcount7        ds        1          ;number of bits sent
rs232Txdivide7       ds        1          ;xmit timing (/16) counter
rs232Txhigh8         ds        1          ;hi byte to transmit
rs232Txlow8          ds        1          ;low byte to transmit
rs232Txcount8        ds        1          ;number of bits sent
rs232Txdivide8       ds        1          ;xmit timing (/16) counter


org   bank3_org

bank3                =         $
rs232RxBank1234      =         $
rs232Rxcount1        ds        1          ;number of bits received
rs232Rxdivide1       ds        1          ;receive timing counter
rs232Rxbyte1         ds        1          ;buffer for incoming byte
rs232byte1           ds        1          ;used by serial routines
rs232Rxcount2        ds        1          ;number of bits received
rs232Rxdivide2       ds        1          ;receive timing counter
rs232Rxbyte2         ds        1          ;buffer for incoming byte
rs232byte2           ds        1          ;used by serial routines
rs232Rxcount3        ds        1          ;number of bits received
rs232Rxdivide3       ds        1          ;receive timing counter
rs232Rxbyte3         ds        1          ;buffer for incoming byte
rs232byte3           ds        1          ;used by serial routines
rs232Rxcount4        ds        1          ;number of bits received
rs232Rxdivide4       ds        1          ;receive timing counter
rs232Rxbyte4         ds        1          ;buffer for incoming byte
rs232byte4           ds        1          ;used by serial routines


org   bank4_org

bank4                =         $
rs232RxBank5678      =         $
rs232Rxcount5        ds        1          ;number of bits received
rs232Rxdivide5       ds        1          ;receive timing counter
rs232Rxbyte5         ds        1          ;buffer for incoming byte
rs232byte5           ds        1          ;used by serial routines
rs232Rxcount6        ds        1          ;number of bits received
rs232Rxdivide6       ds        1          ;receive timing counter
rs232Rxbyte6         ds        1          ;buffer for incoming byte
rs232byte6           ds        1          ;used by serial routines
rs232Rxcount7        ds        1          ;number of bits received
rs232Rxdivide7       ds        1          ;receive timing counter
rs232Rxbyte7         ds        1          ;buffer for incoming byte
rs232byte7           ds        1          ;used by serial routines
rs232Rxcount8        ds        1          ;number of bits received
rs232Rxdivide8       ds        1          ;receive timing counter
```

```
rs232Rxbyte8          ds        1          ;buffer for incoming byte
rs232byte8            ds        1          ;used by serial routines


org   bank5_org

bank5                 =         $
MultiplexBank         =         $
isrMultiplex          ds        1
```

2.3 Initialization Section

This provides the initialization part of the 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,#UARTDividen         ;load TxDivide with UART baud rate
      mov  rs232TxDividen,w
where n = I,2,3,4,5,6,7,8
```

Initialization is required to send the data at the desired baud rate. The value of `UART1divide` specifies the number of times the interrupt has to be serviced before a bit is transmitted. For example, at 9600 baud the value of `UART1divide` is 6, which means that a bit is transmitted once for every six times the respective thread is called.

2.4 Interrupt Section

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

The ISR returns with a "retiw" value of -217 every 4.32 microseconds at an oscillator frequency of 50 MHz.

```
;*************************************************************************************************
           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     ;
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 8 UARTS implemented - UART1 to UART8 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):          rs232TxLow1, rs232TxHigh1, rs232TxCount1
;                                 rs232TxLow2, rs232TxHigh2, rs232TxCount2
;                                 rs232TxLow3, rs232TxHigh3, rs232TxCount3
;                                 rs232TxLow4, rs232TxHigh4, rs232TxCount4
;                                 rs232TxLow5, rs232TxHigh5, rs232TxCount5
;                                 rs232TxLow6, rs232TxHigh6, rs232TxCount6
;                                 rs232TxLow7, rs232TxHigh7, rs232TxCount7
;                                 rs232TxLow8, rs232TxHigh8, rs232TxCount8
;
;     Input Flag(s):              rs232Tx1Flag, rs232Tx2Flag, rs232Tx3Flag, rs232Tx4Flag
;                                 rs232Tx5Flag, rs232Tx6Flag, rs232Tx7Flag, rs232Tx8Flag
;
;     Output variable(s):         rs232Rx1byte, rs232Rx2byte, rs232Rx3byte, rs232Rx4byte
;                                 rs232Rx5byte, rs232Rx6byte, rs232Rx7byte, rs232Rx8byte
;
;     Variable(s) affected :      rs232Txdivide1, rs232Txdivide2, rs232Txdivide3, rs232Txdivide4
;                                 rs232Txdivide5, rs232Txdivide6, rs232Txdivide7, rs232Txdivide8,
;                                 rs232Txcount1, rs232Txcount2, rs232Txcount3, rs232Txcount4
;                                 rs232Txcount5, rs232Txcount6, rs232Txcount7, rs232Txcount8
;                                 rs232Rxdivide1, rs232Rxdivide2, rs232Rxdivide3, rs232Rxdivide4
;                                 rs232Rxdivide5, rs232Rxdivide6, rs232Rxdivide7, rs232Rxdivide8,
;                                 rs232Rxcount1, rs232Rxcount2, rs232Rxcount3, rs232Rxcount4
;                                 rs232Rxcount5, rs232Rxcount6, rs232Rxcount7, rs232Rxcount8
;
;     Flag(s) affected:           rs232Tx1Flag, rs232Tx2Flag, rs232Tx3Flag, rs232Tx4Flag
;                                 rs232Tx5Flag, rs232Tx6Flag, rs232Tx7Flag, rs232Tx8Flag
;                                 rs232Rx1Flag, rs232Rx1Flag, rs232Rx3Flag, rs232Rx4Flag
;                                 rs232Rx5Flag, rs232Rx6Flag, rs232Rx7Flag, rs232Rx8Flag
;
;     Program cycles:             32 worst case for Tx, 33 worst case for Rx
;     Variable Length?            Yes.
;*****************************************************************************************
;--------------------------------------VP: RS232 Transmit---------------------------------------
rs232Transmit1
          _bank      rs232TxBank1234       ; switch to serial register bank
          sb         rs232Tx1Flag          ; Is data there for UART1,
```

```
            jmp         :rs232TxOut1          ; then execute the Tx routine otherwise don't.
            decsz       rs232TxDivide1        ; enter Tx routine until Divide val becomes zero
            jmp         :rs232TxOut1          ; i.e don't enter the Tx routine
            mov         w,#UARTDivide1        ; If Divide val becomes 0 & enters the Tx routine,
                                              ; then again load the
            mov         rs232TxDivide1,w      ; Divide val for not to enter the Tx routine 'Divide'
                                              ; times for next bit
            test        rs232TxCount1         ; If count becomes Zero then also don't enter
            snz                               ;
            jmp         :rs232TxOut1;
                                              ; after all barriers then only it will come here
:txbit      clc                               ; i.e Txflag = hi, Divide=0, count != 0
            rr          rs232TxHigh1          ; right shift Tx data
            rr          rs232TxLow1           ; right shift rs232TxLow which contains start bit
            dec         rs232TxCount1         ; decrement bit counter
            snb         rs232TxLow1.6         ; if the bit in viewing window is hi
            clrb        rs232TxPin1           ; Then make transmit pin lo
            sb          rs232TxLow1.6         ; if the bit in viewing window is lo
            setb        rs232TxPin1           ; Then make transmit pin hi
IFNDEF    sendString                          ; If not sendstring
            test        rs232TxCount1         ; test count
            snz                               ; if zero
            clrb        rs232Tx1Flag          ; then clear the Tx flag & come out
ENDIF
:rs232TxOut1

;*******************************************************************************************
rs232Receive1
            sb          rs232RxPin1           ; get current rx bit
            clc                               ; if bit is zero clear the carry
            snb         rs232RxPin1           ; other wise
            stc                               ; set the carrry
            _bank       rs232RxBank1234
            test        rs232RxCount1         ; test the Rx count
            sz                                ; If zero then only load the Rxcount
            jmp         :rxbit                ; if so, jump ahead
            mov         w,#9                  ; in case start, ready 9 bits
            sc                                ; if not start bit don't load the count
            mov         rs232RxCount1,w       ; it is, so load bit count
            mov         w,#UARTStDelay1       ; ready 1.5 bit periods (50MHz)
            mov         rs232RxDivide1,w      ; load fresh Divide value
:rxbit      decsz       rs232RxDivide1        ; If Divide value is not zero after dec
            jmp         :rs232RxOut1          ; then don't go into Rx routine
            mov         w,#UARTDivide1        ; If yes, load fresh Divide val for next bit
            mov         rs232RxDivide1,w      ;
            dec         rs232RxCount1         ; dec the count
            sz                                ; check for Rxcount value
            rr          rs232RxByte1          ; if zero rotate the buf to save the received bits
            snz                               ; check for Rxcount value
            setb        rs232Rx1Flag          ; if zero set the Rx flag to indicate the
                                              ; complete reception of the byte
:rs232RxOut1
```

Note:The above code implemented for one UART is similar for the remaining 7 UART's. There are 2 UARTS inserted in each isrThread. In isrThread4 the "isrMultiplexer" value is reset to 255 as shown below

```
            mov         isrMultiplex,#255     ; reload isrMultiplex so isrThread1 will be run on the
```

```
                                        ; next interrupt.
          jmp       isrOut              ; 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
IFDEF     SX_28AC
          Mov       w,isrTemp0          ; Restore the mode register value.
          mov       m,w
ENDIF
          mov       w,#-intperiod       ;refresh RTCC on return
                                        ;(RTCC = 216-no of instructions
                                        ;executed in the ISR Routine)
          retiw
;*******************************************************************************************
```

3.0 Baud Rate Generation and Timing

As an example of calculating the parameters which control the timing of the 8-UART Virtual Peripheral, consider transmitting data at 57600 baud with four times oversampling (i.e. a sampling frequency of 230.4 kHz).

Transmission time for 1 bit = 1/57600 seconds

The divide ratio `UARTdivide` for the above example is the sampling rate divided by the baud rate and the number of slots for the 8-UART Virtual Peripheral ISR in the multitasker (i.e. `Num`).

So the formula for `UARTdivide` is:

```
UARTdivide = UARTfs/(UARTbaudrate * Num)
= 230400/(57600 * 4) = 1
```

Therefore, setting `UARTdivide` to 1 results in the desired baud rate. In receive mode, the baud rate is calculated in the same way, except that a constant called `UARTstartdelay` is used to skip over the start bit. This constant is equal to 1.5 times the baud period. Its purpose is to ensure that the bits are sampled near the middle of each pulse. Separate `UARTDivide` and `UARTStDelay` constants are used for each UART (e.g. `UARTStDelay1` is used for UART 1, and `UARTStDelay2` is used for UART 2).

3.1 Circuit Design Procedure

The simplest version of the circuit requires two port pins for transmit and receive. If hardware handshaking is used, additional port lines are required. The hardware interface only requires a driver for converting the voltage level of the signals. The same concept can be used to extend and configure two or more independent UARTs.

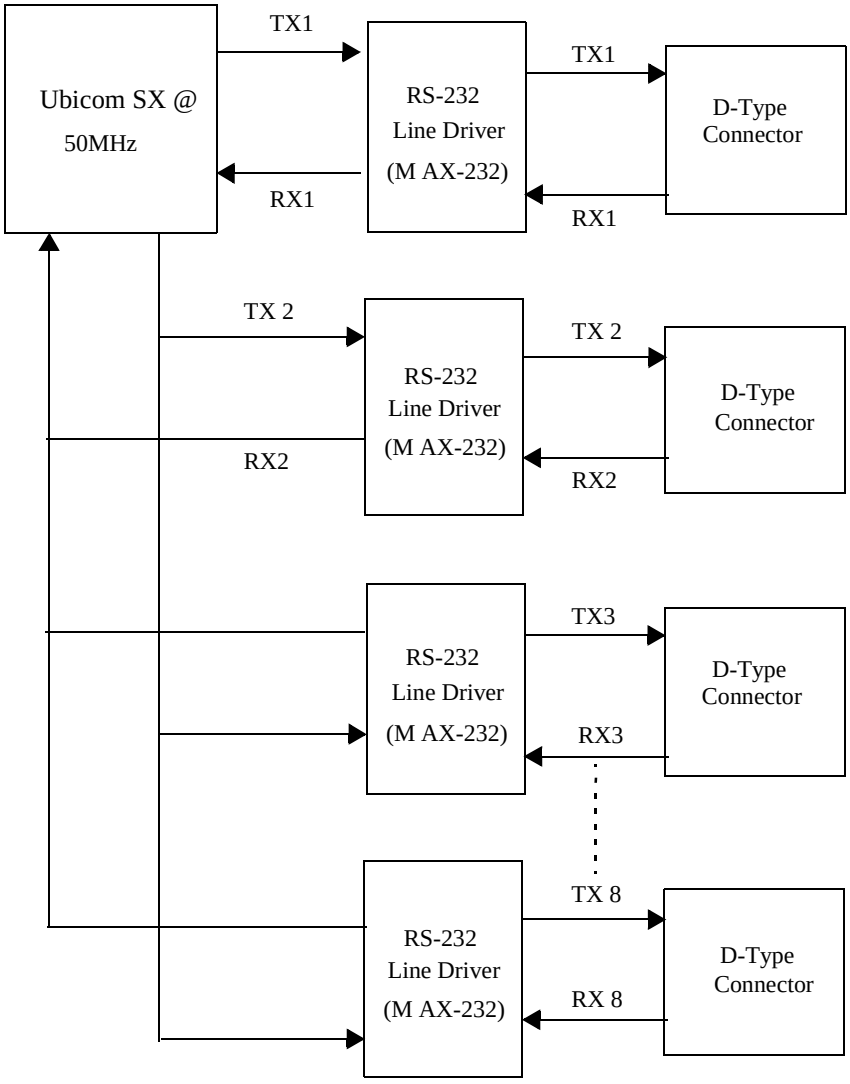

Figure 3-1. Circuit Block Diagram

4.0 Applications

The program is written for a simple UART without hardware handshaking, but it can be modified to include handshaking.

Because this implementation has two UARTs which can be configured for independent baud rates, it can be used in applications communication with two MCUs or peripherals operating at different baud rates.

5.0 TESTING

5.1 Hardware Set Up

- Sx28-52 Demo Board with extra D-Type connector and MAX232 chip.
- Berg pins are provided for the transmit and receive data pins of both connectors.
- Berg pins are also provided for each port pin (i.e. RA2, RA3, RB2, RB3, Rb4, RB5, RB6, RB7, RC0, RC1, RC2, RC3, RC4, RC5, RC6, and RC7), so that all ports can be used alternately with two connectors (one default available on the board and the other wired).
- Out of the 8 UARTs (16 port pins), two UARTs can be tested at a time using the two MAX232 drivers.
- HyperTerminal serial communications program (included with Windows) setup for the UART baud rate.

5.1.1 TEST1

For this test, uncomment `stringTransfer`. This test uses the `sendString` and `getbyte` routines

- In this test, a string stored in the specified location can be sent to eight HyperTerminal applications running on eight PCs using the 8-UART Virtual Peripheral running on the SX28-52 Demo Board. First, the message string “Hit spacebar” is transmitted. To test any UART, connect the corresponding UART transmit and receive port pins to one of the two MAX232 driver pins provided on the board. For example, to send the string through UART 2 and UART 3, connect the transmit (RB3 and RB5) and receive (RB2 and RB4) pins of the UART to the two MAX232 transmit and receive pins. Uncomment the lines `setb rs232TxFlag2` and `setb rs232TxFlag3` in Example 1 of the code, and run the program.

  i.e. RB2 --- Rx1

  RB3 --- Tx1

  RB4 --- Rx2

  RB5 --- Tx2

Observe the message in the respective HyperTerminal windows. Hit space bar to receive the message by uncommenting and calling `call @getByte1` (or the corresponding `getbyte` call for the UART). The message "Yup,The UART works !!!" will be transmitted to the HyperTerminal window connected to the corresponding UART.

5.1.2 TEST2

For this test, uncomment `byteTransfer`. This test uses `getByte()` and `sendByte()` routines using Example 2.

- Get a byte from one HyperTerminal and display it on the same HyperTerminal.

To test UART 2, uncomment `call @getByte2` and `rs232TxFlag2`, and run the program. This test can be run for any UART by uncommenting the respective `call @getByte` and `rs232TxFlag`, and connecting the UART’s port pins to the transmit and receive pins of the MAX232. Bytes can be received from more than one UART at a time by running the respective `call @getByte`, but there is one restriction. If the `getByte1` function is running first, the function is locked until it receives a byte on the corresponding UART. The program will continue to get bytes from the other UART or UARTs only after the `getByte1` function returns.

5.1.3 TEST3

For this test, uncomment `fileTransfer`. This test uses `getByte()` and `sendByte()` routines using Example 3.

- To transfer a text file through UART1 and display it in a HyperTerminal window, uncomment `rs232TxFlag1` and `call @getByte1`, and run the program. To transfer a file using HyperTerminal, use the Transfer tool bar and choose the Send File option, which will prompt you to choose a text file. Using this method, any of the UARTs can be tested by uncommenting their respective `call @getByte` and `rs232TxFlag` functions.

Lit #: AN40-02

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