Eight UART Virtual Peripheral Implementation

Application Note 40

November 2000

1.0 Introduction

The UART Virtual peripheral uses the SX communications controller to provide asynchronous data communication through the RS-232 interface. This Virtual Peripheral enables the SX communications controller 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 MCU's that add functions in the form of additional silicon, the SX Series uses its industry-leading performance to execute functions as software modules, or Virtual Peripheral. These are loaded into a high-speed (20ns access time) 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 UART Virtual Peripheral

The data transmission is performed 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 a line driver.

The 8 UART Virtual Peripheral works simultaneously at different baud rates. As data has to be sent on 8 the UART's simultaneously at different Baud rates it is necessary that the user checks that transmit flag of the particular UART is reset before he sends any data on it, so that data corruption by overwriting of transmit buffer is prevented. As the Virtual Peripheral is configured to send data on all the 8 UART's simultaneously, a significant amount 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 all of the of 8 UART's. If there is data to be received, indicated first by the start flag, a bit of the byte to be received is put into the receive buffer at every pass of the receive ISR routine. Once the complete byte of data(8 bits) is received, a receive flag for the particular UART is set which can be checked in the main loop to pick up the byte from a required UART.

2.1 Program Description

A multithreading concept is used in this Virtual Peripheral to realize the UART. Whenever an RTCC interrupt occurs the program jumps into the interrupt service routine, which contains the interrupt multitasker. The multitasker has a number of threads normally within 24. In the current implementation for the UART Virtual Peripheral, we are using 4 threads and at every occurrence of the interrupt, the interrupt control jumps to one of the threads. Each thread services 2-UART's and each thread executes once every 4 interrupts. 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. This Virtual Peripheral features the capability to send strings that are stored in the area allocated for strings.

Note:In the ISR multithreader, 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 should be changed accordingly.

2.2 Interrupt Service Routine Flowchart for Thread n where n=1,2,3,4

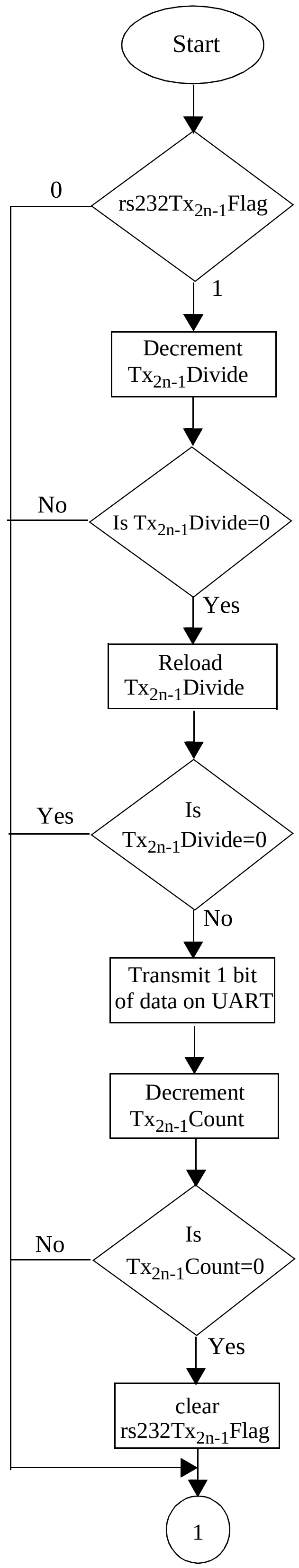

Figure 1. Interrupt Service Routine

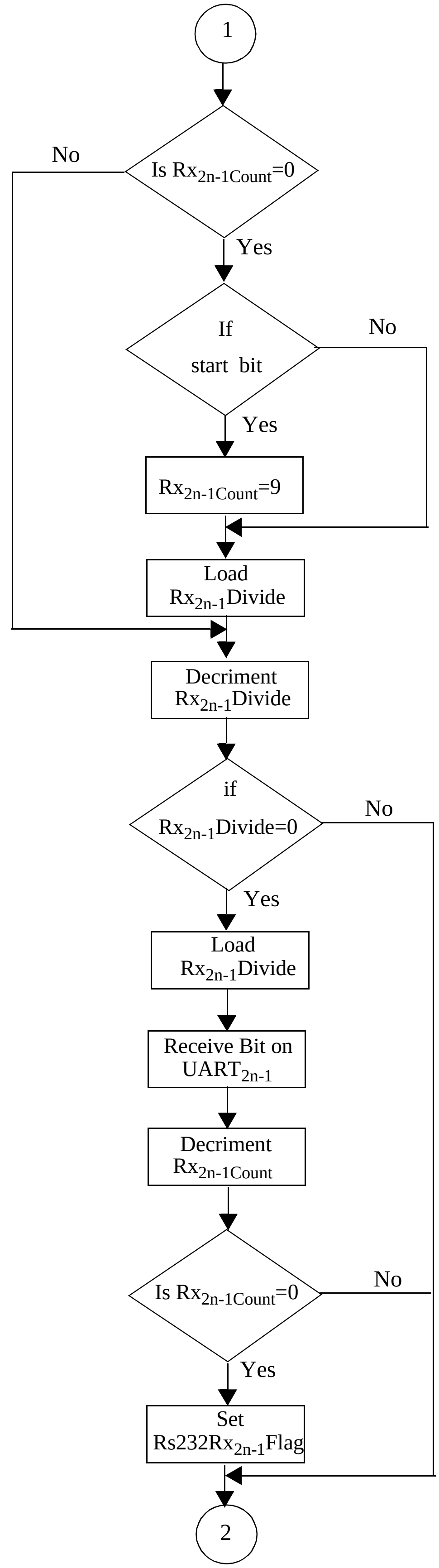

Figure 2-2. Interrupt Service Routine (continued)

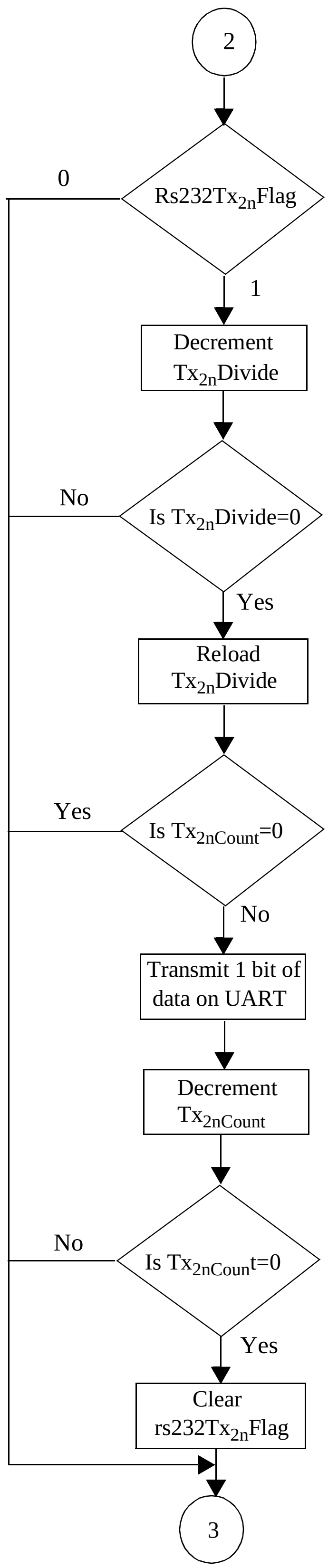

Figure 2-3. Interrupt Service Routine (continued)

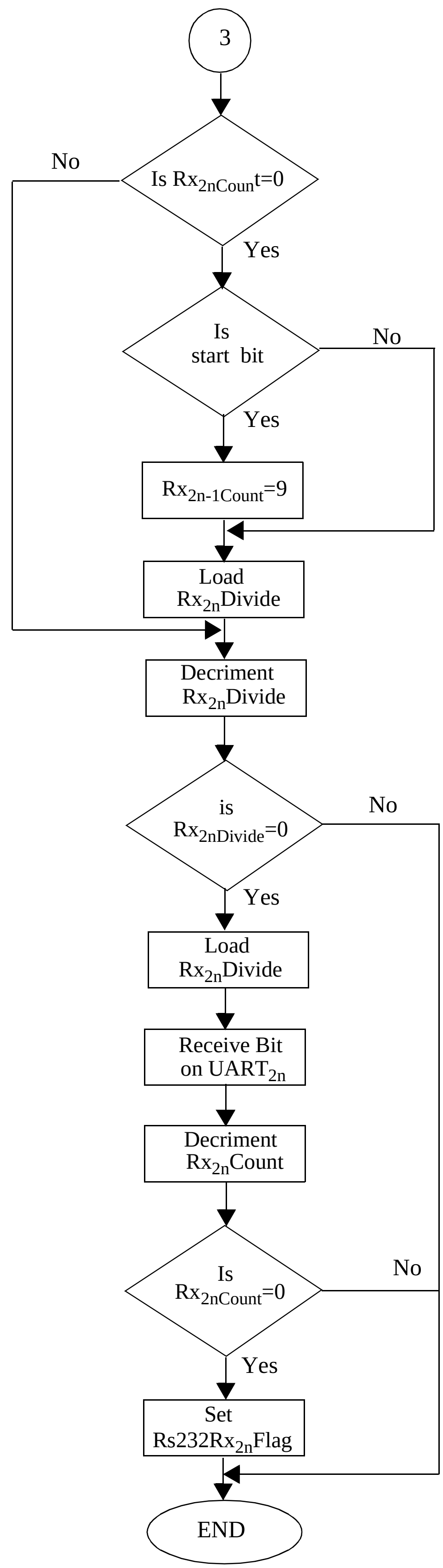

Figure 2-4. Interrupt Service Routine (continued)

3.0 Different Sections of UART Virtual Peripheral

This documentation provides a brief overview of different sections involved in "8-UART Virtual Peripheral Using SX Communications Controller".

The four sections of the 8-UART Virtual Peripheral module mentioned below can be inserted in a main source code at appropriate locations to meet the requirements 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 8 UART Virtual Peripheral module and it also defines the output pins for the 8 UART Virtual Peripheral. The value of UARTDivide, UARTStDelay and pin declarations are made here.

The values of the constants are as follows:

```
UARTfs        =      230400
Num           =      4
Int Period    =      217
```

UARTDividen = UARTfs/(UARTBaudn * Num)

UARTStDelayn = UARTDividen + (UARTDividen/2)+1

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

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, Rb 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      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 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 ' uart1baud1920 ' is uncommented it implies that UART-1 has a baud rate of 19200bps, similarly ' uart2baud9600 ' implies UART-2 is to be configured for a baud rate of 9600bps.

3.2 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
```

3.3 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
```

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 the rate of 9600bps, the value of UARTDivide is 6, this means that every one bit should be transmitted once in 6 times of the occurrence of the respective isrThread.

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         ;
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
;************************************************************************************************
```

4.0 Features

4.1 Baud Rate Generation Methodology and Timing

To understand the method used 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 = 1/57600 sec

Data is sampled at a frequency of 4 * 57,600 bps = 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 UARTStDaley constants are computed for feasibility of all the UART's being independent of each other.

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 connected to the driver directly which takes care of the level conversion.

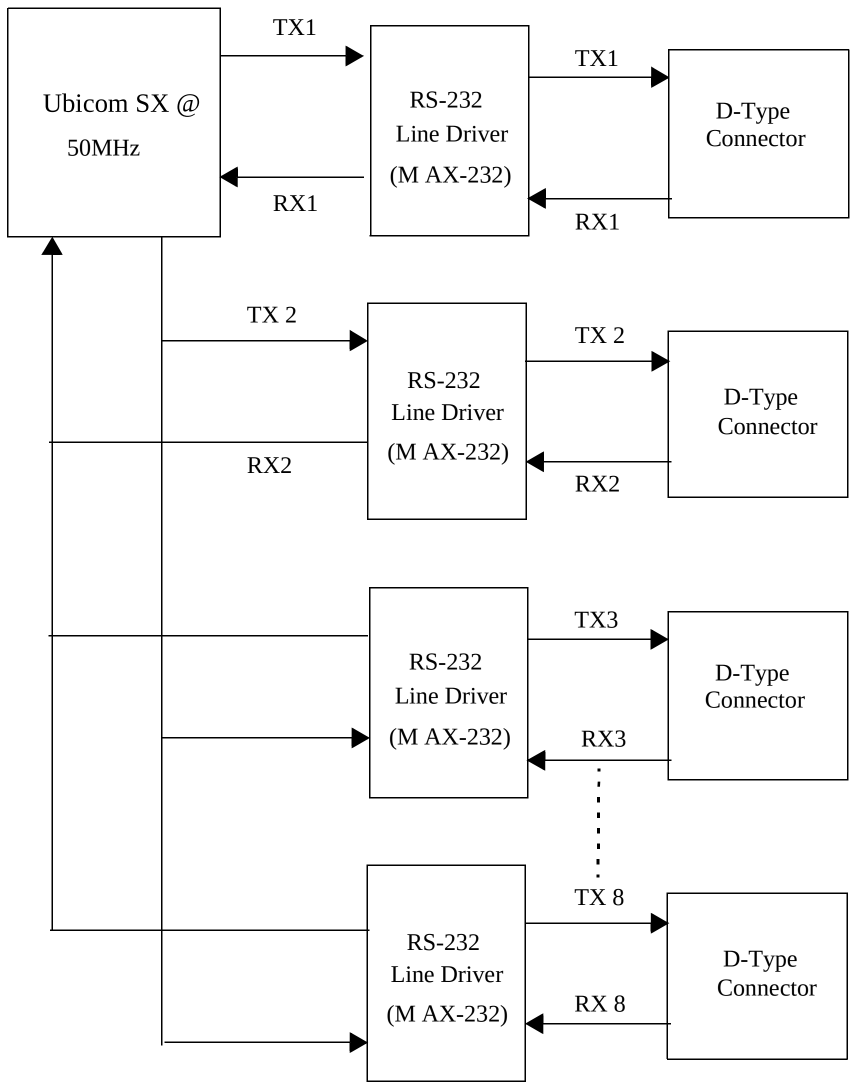

Figure 4-1. Circuit Block Diagram

5.0 Applications

The Applications of UART are innumerable and the use of UART is reflected in nearly every communications 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 8 UART's with no handshaking. The program can be modified to handle handshaking as well.

As this implementation contains 8 UART's which can be configured for different baud rates, it is possible to send messages quickly as 8 UARTs are independent of each other. Hence it can be used in Applications where we have 8 MCU's or peripherals operating at different baud rates.

6.0 TESTING

6.1 Hardware Set up Required for Testing

- Sx28-52 Demo Board with extra D -Type connector and MAX232 chip.
- Berg pins are provided for Tx-pin & Rx-pin of both connectors.
- Berg pins are also provided for each port pins (i.e. RA2, RA3, RB2, RB3, Rb4, RB5, RB6, RB7, RC0, RC1, RC2, RC3, RC4, RC5, RC6, RC7), so that we can use all ports alternatively with two connectors (one default available on the board & the other wired).
- Out of the 8 UART's (16 port pins), we can test 2 UART's at a time using the 2 MAX232 drivers.
- Hyper terminal setup as per the required baud rate of the UART used.

6.1.1 TEST1

For this test uncomment the "stringTransfer". In this test will use "sendString" and "getbyte" routines using Example 1.

- In this test, string stored in the specified location can be sent to 8 hyper terminal applications running on 8 PC's using the 8 UART Virtual Peripheral modules on the SX28-52 Demo Board. First the message string ' Hit spacebar ' is transmitted on the UART's. If you want to test any UART, then connect corresponding UART port pins Tx & Rx pins to one of the 2 MAX232 driver pins provided on the board. For example, you want to send string through UART 2 & 3 then connect the Tx (RB3 & RB5) and Rx (RB2 & RB4) pins of the UART to the two MAX232's Tx & Rx pins. Uncomment lines "setb rs232TxFlag2" and "setb rs232TxFlag3" in Example 1 of the code. Run the program.

  i.e. RB2 --- Rx1

  RB3 --- Tx1

  RB4 --- Rx2

  RB5 --- Tx2

Observe the send message on the respective Hyperterminals. If you hit space bar on the hyper terminal, then the respective UART receive the message by running getByte1() (or the corrsponding getbyte for the UART) for which we need to un-comment "call @getByte1" (or the corrsponding call @getbyte for the UART). The message " Yup,The UART works !!!" will be transmitted to the Hyper terminal of the corresponding UART.

6.1.2 TEST2

For this test uncomment "byteTransfer". In this test we use "getByte()" and "sendByte()" routines using Example 2.

- Get a byte from one hyper terminal and display same on the same hyper terminal.

If you want to test this for UART2 then uncomment "call @getByte2" and "rs232TxFlag2", and run the program. This test can be run for all the UART's by un-commenting the respective "call @getByte" and "rs232TxFlag" of the UART, and connecting respective port pins to Rx & Tx pins of MAX232. We can get a byte from more than one UART at a time by running the respective call @getByte. But there is one restriction, suppose getByte1 function is running first then the function is locked until it receives a byte on the corresponding UART. Only after the getByte1 returns, the program will continue to get a byte from the other UART or UART's.

6.1.3 TEST3

For this test uncomment "fileTransfer". In this test we use "getByte()" and "sendByte()" routines using Example 3.

- If you want to transfer a text file through UART1 and display the same on the hyper terminal, then uncomment "rs232TxFlag1" and "call @getByte1". Then run the program. To transfer a file using the hyper terminal, use the "Transfer" tool bar and choose "Send File" option, which will prompt you to choose a text file. Using the same method, all the UART's can be tested by uncommenting the respective "call @getByte" and "rs232TxFlag"s.

Lit #: AN40-01

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