Asynchronous Transmitter & Receiver (UART) Virtual Peripheral Implementation

Application Note 38

September 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 makes the SX device act as Universal Asynchronous Transmitter and Receiver and communicate with any PC directly. 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 modules, 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 Peripheral modules 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 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 level through the line driver.

2.1 Program Description

A multithreading concept is used to realize the UART Virtual Peripheral module. Whenever an 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 UART Virtual Peripheral, 16 threads are used at every occurrence of the interrupt. The interrupt control jumps to one of the threads. The Virtual Peripheral modules are inserted into one of the threads. The UART module is contained within isrThread1. This thread is executed every 4th interrupt. Therefore the UART routine executes once for the occurrence of 4 interrupts. This technique enables the user to embed other Virtual Peripheral modules within the remaining threads.

2.2 Interrupt Service Routine Flowchart

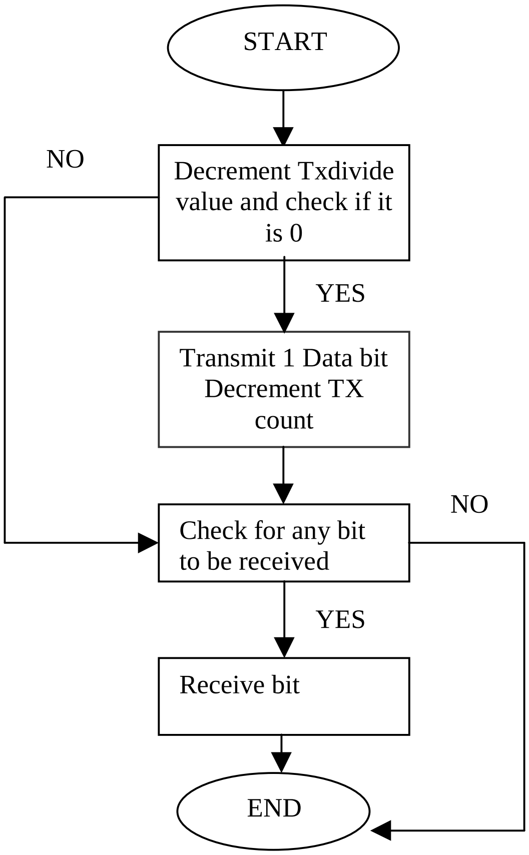
START
NO
Decrement Txdivide value and check if it is 0
YES
Transmit 1 Data bit Decrement TX count
NO
Check for any bit to be received
YES
Receive bit
END

Figure 2-1. Interrupt Service Routine Flowchart

Ubicom™ and the Ubicom logo are trademarks of Ubicom, Inc.
All other trademarks mentioned in this document are property of their respective componies.

3.0 UART Virtual Peripheral Building Blocks

There are four sections associated with the of the UART Virtual Peripheral. Each section can be inserted in a main source code at appropriate locations to meet the requirement of realizing the UART Virtual Peripheral.

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

3.1 Equates Section

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

```
UARTdivide = UARTfs/(UARTbaud * Num)
UARTStDelay = UARTDivide + (UARTDivide/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 is used for the external interface.

The Pins are configured as follows:

• Ra.0	rs232RTSpin
• Ra.1	rs232CTSpin
• Ra.2	rs232Rxpin
• Ra.3	rs232Txpin

```
intPeriod          = 217
UARTfs             = 230400
Num                = 4

IFDEF baud1200
UARTBaud           = 1200
ENDIF

IFDEF baud2400
UARTBaud           = 2400
ENDIF

IFDEF baud4800
UARTBaud           = 4800
ENDIF

IFDEF baud9600
UARTBaud           = 9600
ENDIF

IFDEF baud1920
UARTBaud           = 19200
ENDIF

IFDEF baud5760
UARTBaud           = 57600
ENDIF
```

3.2 Bank Section

This section describes the use of the banks in the UART Virtual Peripheral. The bank used in the UART Virtual Peripheral module (BANK 1) should be same in the main source template, if used with other Virtual Peripheral modules.

Inside this bank we have different banks for RS232TX, RS232RX and Multiplex just for clarity. Though there are three banks, as all the three are declared within bank1, it will point to bank 1 whenever any of these three banks are accessed.

```
Org          bank1_org
; VP: VP Begin          RS232 Transmit

rs232TxBank                =            $             ;UART bank
rs232Txhigh                ds           1             ;hi byte to transmit
rs232Txlow                 ds           1             ;low byte to transmit
rs232Txcount               ds           1             ;number of bits sent
rs232Txdivide              ds           1             ;x’mit timing (/16) counter
Rs232Txflag                ds           1
;VP: END

;VP : VP Begin          RS232 Receive

rs232RxBank                =            $
rs232Rxcount               ds           1             ;number of bits received
rs232Rxdivide              ds           1             ;receive timing counter
rs232Rxbyte                ds           1             ;buffer for incoming byte
rs232byte                  ds           1             ;used by serial routines
hex                        ds           1
;VP: END
;VP : VP Begin          Multiplexer

MultiplexBank              =            $
isrMultiplex               ds           1
;VP: END
```

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,#UART1divide   ;load Txdivide
                            ;with UART baud
                            ;rate
mov        rs232TXdivide,w
```

This initialization is done to send the data at the required baud rate. The value of UART divide 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 UART divide is 6, this means that every one bit should be transmitted once in 6 times of the occurrence of thread1.

3.4 Interrupt Section

It provides the interrupt service routine for the UART Virtual Peripheral. The flow of the interrupt service routine is demonsrated by the flowchart in Figure 4-1.

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

The algorithm of the Interrupt is as follows:

- An interrupt occurs whenever the RTCC register rolls from FF to 00 value.
- On the Occurrence of an Interrupt, the control flows to the interrupt routine written at 00
- location
- In the ISR the value of "isrmultiplexer" register is incremented, initially it is 0
- This is added to the value of the program counter to jump to the respective ISR thread.
- In the "isrThread1" the UART Virtual Peripheral is inserted so every time the control goes to Isrthread1 the Virtual Peripheral is executed.
- The value of TXdivide is checked for zero, to confirm whether a bit has to be transmitted in this cycle.
- The Value of UARTdivide is loaded to Txdivide.
- The value of TX count is checked to confirm the presence of data to be transmitted
- If the value is not zero, there is data to be transmitted, then the transmit routine is executed.
- The data stored in Txhigh register is pushed to w register.
- The MSB of the TX low register is set, this is the start bit. A total of ten bits are transmitted which consists of 1 start bit+8 databits + I stop bit. The receiving side terminal has to be configured for these values with the baud rate required for efficient working of the UART
- The bits are rotated to the right and fed to the TX low register.
- The bit 6 of the TX low register is transmitted on the TX line.

A Similar procedure is adopted to receive the incoming bytes.

The code of the interrupt service routine is given below:

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

;*******************************************************************************
;------------------------VP:VP Multitasker--------------------------------------
;*******************************************************************************
; Virtual Peripheral Multitasker up to 16 individual threads, each running at
; the(interrupt rate/16).
; Input variable(s):isrmultiplex  : variable used to choose threads
; Output variable(s): None executes the next thread
; Variable(s) affected: isr_multiplex
; 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
;---------------------------VP: RS232 Transmit----------------------------------
;*******************************************************************************;
; 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.

; RECEIVING: The rs232Rxflag is 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, if not needed,
; to ;reduce execution time and memory usage, as long as the initial "BANK serial" (common)
; instruction is kept.
; Input variable(s) : rs232Txlow (only high bit used), rs232Txhigh, rs232Txcount .
; output variable(s) : rs232Rxflag, rs232Rxbyte
; variable(s) affected : rs232Txdivide, rs232Rxdivide, rs232Rxcount
; Flag(s) affected : rs232Rxflag
; Program cycles: 17 worst case
; Variable Length?  Yes.
;*******************************************************************************
rs232Transmit
       _bank        rs232TxBank         ;2 switch to serial register bank
       decsz        rs232Txdivide       ;1 only execute the transmit routine
       jmp          :rs232TxOut         ;1
       mov          w,#UARTdivide       ;1 load UART baud rate (50MHz)
       mov          rs232Txdivide,w     ;1
       test         rs232Txcount        ;1 are we sending?
       snz                              ;1
       jmp          :rs232TxOut         ;1
:txbit clc                              ;1 yes, ready stop bit
       rr           rs232Txhigh         ;1 and shift to next bit
       rr           rs232Txlow          ;1
       dec          rs232Txcount        ;1 decrement bit counter
       snb          rs232Txlow.6        ;1 output next bit
       clrb         rs232TxPin          ;1
       sb           rs232Txlow.6        ;1
       setb         rs232TxPin          ;1,17

:rs232TxOut
```

```
;---------------------------VP: RS232 Receive-------------------------------------
;**********************************************************************************
; 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. RECEIVING: The rx_flag is 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.
; Output variable(s) : rx_flag, rx_byte
; Variable(s) affected : tx_divide, rx_divide, rx_count
; Flag(s) affected : rx_flag
; Program cycles: 23 worst case
; Variable Length?  Yes.
;********************************************************************************

rs232Receive
             _bank         rs232RxBank                ;2
             sb            rs232RxPin                 ;1 get current rx bit
             clc                                      ;1
             snb           rs232RxPin                 ;1
             stc                                      ;1
             test          rs232Rxcount               ;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           rs232Rxcount,w             ;1 it is, so renew bit count
             mov           w,#UART1startdelay         ;1 ready 1.5 bit periods (50MHz)
             mov           rs232Rxdivide,w            ;1
:rxbit       decsz         rs232Rxdivide              ;1 middle of next bit?
             jmp           :rs232RxOut                ;1
             mov           w,#UARTdivide              ;1 yes, ready 1 bit period (50MHz)
             mov           rs232Rxdivide,w            ;1
             dec           rs232Rxcount               ;1 last bit?
             sz                                       ;1 if not?
             rr            rs232Rxbyte                ;1 then save bit
             snz                                      ;1 if so,
             setb          rs232RxFlag                ;1,23 then set flag
:rs232RxOut
             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 reload isrMultiplex so isrThread1 will be
; run on the next interrupt. This thread must reload the isrMultiplex register since it is
; the last one to run in a rotation.

             _bank      Multiplexbank
             mov        isrMultiplex,#255
             jmp        isrOut                      ; 7 cycles until mainline program resumes
                                                    ; execution

                        isrOut
; Set Interrupt Rate
                        Isrend


; refresh RTCC on return (RTCC = 217-no of instructions executed in the ISR)

             mov        w,# -intperiod
             retiw                                  ;return from the interrupt

; End of the Interrupt Service Routine
```

4.0 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

Time taken for the transmission of 1 bit of data = 1/57600 sec

As data is sampled at a frequency of 230.4KHz, time taken to send I bit = 1/230400 sec

If data is sent at the sample rate the 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 times the occurrence of the interrupt, the baud rate matching will be taken care.

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 id being called 4 times in the Interrupt Multitasker hence the value of UART divide is further divided by 4 to get a resulting value of 1.

So the formula for UART divide will be:

UARTdivide = UARTfs/(UARTbaudrate*number of times the ISR is called in the Multitasker)

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

In the receiving mode the generation of baud rate is in the same way as explained above.

But a constant called UARTstartdelay is introduced which is equal to 1.5 times the bit length is just to take care for the start bit as it is not used.

4.1 CIRCUIT DESIGN PROCEDURE

The simplest version of the circuit requires two SX pins 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. The same concept can be used to extend and configure 2 independent UART’s or Multiple UART’s.

Lit #: AN38-02

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