Universal Asynchronous Transmitter/Receiver (UART) Virtual Peripheral

Application Note 38

June 2001

1.0 Introduction

The UART Virtual Peripheral™ uses the SX communications controller to provide asynchronous serial communication through an RS-232 interface. The Virtual Peripheral provides direct communication with any device that has an RS-232 interface, such as a PC. 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.

2.0 Program Description

The UART Virtual Peripheral is designed to operate at at a fixed frequency driven by periodic interrupts. In the SX, the mechanism used to generate periodic interrupts of a fixed frequency is the RTCC timer. The UART virtual peripheral can be run as one of several threads in a multithreaded interrupt service routine (ISR) or as part of a single thread. The code this document describes contains a multi-tasking ISR. 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*. The UART Virtual Peripheral is split into a transmit section and a receive section, and each of these sections is assigned to `isrThread1`. In this example, the multitasker implements 16 slots for calling threads, and four of these slots are occupied by calls to `isrThread1`. Because the UART Virtual Peripheral only receives service on every fourth interrupt, most of the CPU bandwidth is available for use by other Virtual Peripherals.

2.1 Interrupt Service Routine Flowchart

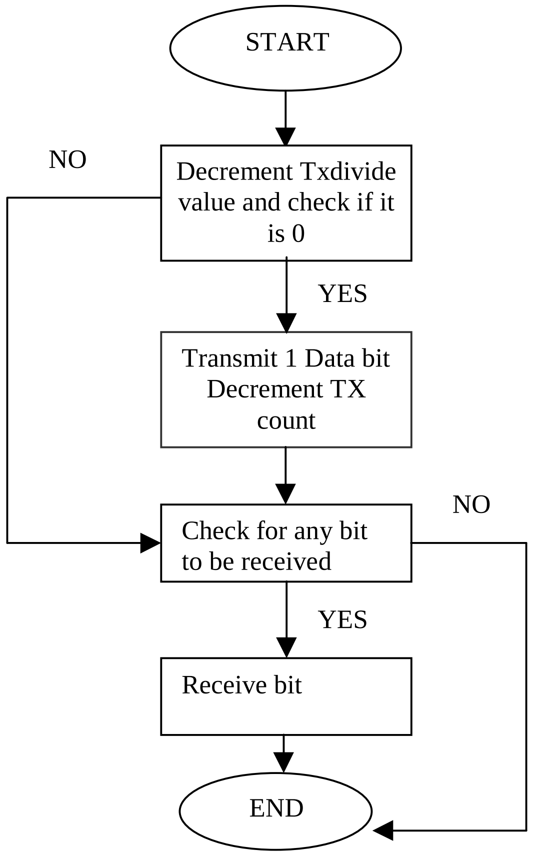
START
Decrement Txdivide value and check if it is 0
NO
YES
Transmit 1 Data bit Decrement TX count
Check for any bit to be received
NO
YES
Receive bit
END

Figure 2-1. Interrupt Service Routine Flowchart

3.0 Source Code Sections

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

- Equates Section: Constants that define the operation of the UART
- Bank Section: Memory allocation for the virtual peripheral
- Initialization Section: Must run before the mainline application code can run
- Interrupt Section: The most important part of the virtual peripheral; the section of code that continually runs (while enabled) and mimics hardware in software
- API Subroutines: Send data to and receive data from the Virtual Peripherals. Analogous to the API routines that would be written for accessing hardware, and make the operation of the virtual peripheral transparent.

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.

3.1 Equates Section

The equates section provides configuration constants, allowing compile-time changes in the functionality of the UARTs.

`UARTRxFs`, `UARTTxFs`: These constants define the SAMPLING RATE of the UART, and must be calculated by the software programmer based on the Interrupt Frequency and on how the `ISR-Multitasker` is set-up. In this version of the source code, the UART transmitter and receiver are both sampled at rates of 57600Hz.

$$\text{Interrupt Frequency} = \frac{Oscillator\ Frequency}{IntPeriod}$$

$$\texttt{UARTTxFs} = interrupt\ frequency * \frac{number\ threads\ calling\ rs232Transmit}{total\ number\ of\ ISR\ threads}$$

By Default,

oscillator frequency = 50MHz

Interrupt frequency » 230400Hz

Number of threads calling the `rs232Transmit/Receive routines` = 4

Number of ISR Threads is 16.

Therefore, `UARTTxFs` and `UARTRxFs` = 57600.

The multitasker rotates interrupt service among 16 slots, and `isrThread1` is called from four of these slots. In other examples, one slot of 16 might be sufficient to service the UART Virtual Peripheral ISR.

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

The port 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 register banks in the UART Virtual Peripheral.

Inside this bank we have different banks for RS232TX, RS232RX, and Multiplex for clarity. Although there are three banks, because all three are declared within bank 1, bank 1 becomes the current bank whenever any of these three banks are accessed. Using different defines for each module allows the RAM definitions to be moved around in the future to make space for other Virtual Peripherals in the same application.

```
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 section provides initialization for the UART Virtual Peripheral. This has to be included with the initialization of all other ports and registers, prior to entering the main loop of the application.

```
_bank      rs232TxBank      ;select rs232 bank
mov        w,#UARTTxDivide  ;load Txdivide
                            ;with UART baud
                            ;rate
mov        rs232TxDivide,w
```

Initialization is required to send the data at the desired baud rate. The value of `UARTTxDivide` specifies the number of times the thread has to be serviced before a bit is transmitted. For example, at 9600 baud the value of `UARTTxDivide` is 6 (57600/9600=6), which means that a bit is transmitted once for every six times `isrThread1` is called.

3.4 Interrupt Section

This section is the UART Virtual Peripheral ISR. The flow of the interrupt service routine is demonstrated by the flowchart in Figure 2-1.

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

The events that occur on an interrupt are:

1. An interrupt occurs whenever the RTCC value rolls from $FF to $00. The interrupt jumps to the ISR at $00.
2. In the ISR, the value of `isrMultiplex` is incremented. Initially it is zero. `isrMultiplex` implements rotation among the slots for threads called by the multitasker. `isrMultiplex` is added to the value of the program counter to jump into a table of jump instructions for the threads.
3. In `isrThread1`, the UART Virtual Peripheral ISR is executed.
4. The value of `rs232TxDivide` is checked for zero, to confirm whether a bit has to be transmitted in this cycle. The value of `UARTTxDivide` is loaded to `rs232TxDivide`.
5. The value of `rs232TxCount` is checked to confirm the presence of data to be transmitted. If the value is not zero, there is data to be transmitted, in which case the transmit routine is executed.
6. The data stored in `rs232TxHigh` is pushed to the W register.
7. The MSB of the `rs232TxLow` register is set, which is the start bit. A total of ten bits are transmitted, which consists of 1 start bit + 8 data bits + 1 stop bit. The receiver has to be configured for this format.
8. The bits are rotated to the right and fed to the `rs232TxLow` register.
9. Bit 6 of the `rs232TxLow` register is transmitted on the TX pin.

A similar procedure is used to receive the incoming bytes.

The source code of the interrupt service routine is shown 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,#UARTTxDivide     ;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,#UARTRxStDelay          ;1 ready 1.5 bit periods (50MHz)
             mov          rs232RxDivide,w           ;1
:rxbit       decsz        rs232RxDivide             ;1 middle of next bit?
             jmp          :rs232RxOut               ;1
             mov          w,#UARTRxDivide           ;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 and Timing

As an example of calculating the parameters which control the timing of the UART Virtual Peripheral, consider transmitting data at 14400 baud with four times oversampling (i.e. a sampling frequency of 57.6 kHz).

Transmission time for 1 bit = 1/14400 seconds

The divide ratio `UARTTxDivide` for the above example is the sampling rate divided by the baud rate. The sampling rate is defined by the interrupt frequency and the number of slots for the UART virtual peripheral ISR in the multitasker.

So the formula for the `UARTTxDivide` and `UARTRxDivide constants` is:

```
UARTTxDivide = UARTTxFs/(UARTBaud)
= 57600/(14400 * 4) = 1
```

In receive mode, the generation of baud rate is calculated in the same way, except that a constant called `UARTRxStDelay` 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.

4.1 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.

Lit #: AN38-03

Sales and Tech Support Contact Information

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

UBICOM™

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