Interrupt Driven RS232 Receive and Transmit

This is my fully buffered, fully interrupt driven RS232 module for PIC16F877. It is does not waste any time with sending characters and waiting for them to go (I hate routines like that!). With this set you can Tx by filling a buffer and then set the Tx running and just get on with your application - the interrupt will take care of everything and stop once the buffer is empty.

It relies on some macros which you'll find elsewhere in the library.

Enjoy

Preamble

```
        ; originally for PIC16F877 - you should be able to adapt it without
too much hassle

    ; *** Bank0/1/2/3 mirrored in all banks 0x70, 0xF0, 0x170, 0x1F0, 16
bytes
    ;  accesible from all banks

        CBLOCK        0x70
            GENTEMP        ; 0
            FLAGS        ; 3 application and subsystem flags
                        ;   - 0 RX Buffer active - we have stuff in the
buffer
                        ;   - 1 RX Buffer OVF - the buffer has overflowed
                        ;   - 2 RS232 TXINPROGRESS flag - we must wait to
put a char in the buffer if set
                        ;   - 3 RX Buffer full - no more space - next char
will overflow
                        ;   - 4 RX buffer has recieved a <cr>
                        ;   - 5
                        ;   - 6
                        ;   - 7
            RXCHTEMP    ; 4
            TXTEMPFSR    ; 5     |
            TXTEMPSTATB    ; 6  |
            TXCHTEMP    ; 7  | context saving in TXBUFFQ
            TXTEMPSTAT    ; 8 /
            SAVED_W            ; 9  |
            SAVED_STATUS    ; A  | context saving in ISR
            SAVED_PCLATH    ; B  |
            SAVED_FSR    ; C /

        ENDC

    ; *** Bank1 *** 80 bytes
         CBLOCK         0xA0
             RXSHUFFSRC
             RXSHUFFDST
             RXBUFFRDPTR
             RXBUFFWRPTR
            RXBUFF:RXBUFFSIZE
         ENDC

    ; *** Bank2 *** extra ram 96 bytes
        CBLOCK    0x110
            TXBUFFPTR        ; when TXINPROGRESS=0; points to the free place in the buffer for TXBUFFQ
                        ; when TXINPROGRESS=1; used to step through the buffer by TXBUFFUNQ
             TXBUFF: TXBUFFSIZE
                         ; this buffer is a good size general purpose text buffer. Although it is aimed
                         ; at RS232 TX, it can be used to hold strings for any reason
                         ; binary and bcd outputs write here but the output doesn't go anywhere
                         ; until we say (or the buffer overflows)
        ENDC

    ; *** Bank3 *** extra ram 96 bytes
        CBLOCK        0x190
            CTR,II            ; gp counters
            TF,TF2            ; pointers
        ENDC
```

ISR considerations

```

ISR:
        ORG    4

        PUSH

    ;BANK0 is implicit from the PUSH macro

    ; the handler routines are arranged in order of urgency

ISR_RX_IRQ: ; RS232 Rx char recieved
        SKIPHI        PIR1,5
        GOTO        ISR_RX_IRQRET
        LO        PIR1,5            ; clear flag
        MOVFW        RCREG
        GOTO        RXBUFFQ                    ; in the RS232 module
ISR_RX_IRQRET:

ISR_TX_IRQ: ; RS232 Tx Complete
        SKIPHI        PIR1,4
        GOTO        ISR_TX_IRQRET
        CALL         TXBUFFUNQ
ISR_TX_IRQRET:


ISREND:
        POP

        RETFIE
```

the actual RS232 routines

```

;
; RS232 Module
; Routines:
;       TXBUFFQ            Place the char in W in the buffer but doesn't
send anything. If you fill the buffer, it will trigger TXSTART
;                                       and you'll be kept waiting while the
buffer empties, then your char is put in the buffer for next time.
;       TXBUFFUNQ        only called as part of the Tx ISR! ***do not
call*** Sets the Tx flag and so empties the buffer to the RS232 TX line in
the background.
;       TXSTART            Start TXBUFFUNQ - set the flag to begin
outputting chars from the buffer - usually causes an immediate interrupt
(because of TXEN=1)
;                                       chars must be buffered. To output a
single char immediately:
;                                       MOVLW   "*"        - my character
;                                       CALL    TXBUFFQ            -
effectively a "print W" routine
;                                       CALL    TXSTART         - char will
be output as part of the buffer
;
;       RXBUFFQ            This is the ISR handler for RX - places the Rx
byte in the buffer
;       RXBUFFREAD        Read a character from the buffer if there is one;
returns W=0 if not
;       RXBUFFSHUFFLE            Remove read chars from the buffer
;       RXBUFFCLEAR        Clear the buffer and reset all pointers & flags
;
;
; if the buffer fills during TXBUFFQ, TXSTART is called implicitly. Thus
TXBUFFQ can *always* take your char
; but you might have to wait for the buffer to empty. Cannot buffer chars
while sending - yet!
; buffer is empty after TXBUFFUNQ
;
; FLAGS,2 is a global "TX in progress flag"
;
; has specific register requirements - see the kernel

```

```
TXBUFFSIZE  EQU    D'80'+1        ;+1 allows full buffer size plus the zero endstop
RXBUFFSIZE  EQU     D'40'

    #DEFINE        TXINPROGRESS    FLAGS,2    ; TX Buffer is being emptied - no more queuing until finished

    #DEFINE        RXBUFFACTIVE    FLAGS,0    ; RX Buffer active - we have stuff in the buffer
    #DEFINE        RXBUFFEROVF        FLAGS,1    ; RX Buffer OVF - the buffer has overflowed - the data is unreliable because chars have been lost
    #DEFINE        RXBUFFERFULL    FLAGS,3    ; RX Buffer FULL - next char will cause overflow

;*************************************
;       INSIDE THE ISR!!!!
;*************************************
; un-queue the next character in the buffer. Buffer must end with zero byte
; if the buffer is empty (we don't want any more interrupts), ensure we have
; finished sending the last byte and disable the Tx and thus its interrupt.

TXBUFFUNQ:
        BANK2                    ; all TX Buffers & ptrs are in BANK2, don't use quick banks coz of INDF
        MOVLF        LOW TXBUFF,FSR            ; calculate the current character position in the buffer
        MOVFW        TXBUFFPTR
        ADDWF        FSR            ; here INDF is the nth charctaer in the buffer
        MOVFW        INDF
        JMPZ        NOCHARS            ; end of the data?
        BANK0F                    ; quick bank0    :
        MOVWF        TXREG
        BANK2F                    ; quick bank2    :
        INCF        TXBUFFPTR        ; ... and increment the pointer for the next char
TIDYEXIT:
        BANK0F
        RETURN

; we have a char zero - we are at the end of the data or have nothing to send.
; We interrupted (we are here) so we need to disable TXEN but not until
; TRMT goes high
TXBUFFCLR:
NOCHARS:
        CLRF        TXBUFFPTR        ; clear the Tx buffer: reset the pointer to 0...
        CLRF        TXBUFF            ; ... and clear the fisrt byte in the buffer
        BANK1
        BTFSS        TXSTA,TRMT        ; check if the last character has
finished sending
        GOTO        TIDYEXIT        ; if not, just exit
        LO        TXSTA,TXEN        ; We finished sending so disable the Tx
to remove the interrupt
        LO        TXINPROGRESS            ; tell the world we are no longer
emptying the buffer
        GOTO         TIDYEXIT        ; and play nicely



; the RX buffer routine
; jumped-to from the ISR RX handler so consider it in the ISR
; W contains the recieved char
RXBUFFQ:
        BANK1
        BTFSC        RXBUFFERFULL            ; the buffer has space ?
        GOTO        RXBUFFBROKE
        MOVWF        RXCHTEMP        ; save the char
        CP        RXCHTEMP,D'13'
        BTFSC        STATUS,Z
        HI        FLAGS,4            ; current character is a <cr>
        MOVLW        LOW RXBUFF        ; point to the start of the buffer...
        ADDWF        RXBUFFWRPTR,W            ; add the pointer
        MOVWF         FSR            ; here INDF is the nth character in the
buffer
        MOVFF        RXCHTEMP,INDF            ; put the char in the buffer
        INCF        RXBUFFWRPTR        ; move the pointer along
        HI        RXBUFFACTIVE            ; signal we have stuff

        CP        RXBUFFWRPTR,RXBUFFSIZE    ; check for end of buffer
        SKIPNZ                    ; recieve buffer is not full
        HI        RXBUFFERFULL            ; you need to take some stuff out
of the buffer immediately
        GOTO        TIDYEXIT

RXBUFFBROKE:
        HI        RXBUFFEROVF        ; oh dear... got a char but no room for
it
        GOTO        TIDYEXIT



;*************************************
;       OUTSIDE THE ISR!!!!
;*************************************
; this starts the TX buffer emptying. It does this by simply enabling TXIF
; then everything is handed off to the ISR.
TXSTART:
        BTFSC        TXINPROGRESS            ; jump back if we are already
doing it
        RETURN
        DI
        MOVFF        STATUS,TXTEMPSTATB    ; preserve the bank bits
        HI        TXINPROGRESS            ; tell the world we are emptying
the buffer
        BANK2
        CLRF        TXBUFFPTR        ; this pointer is used to empty the
buffer now
        BANK1
        HI        TXSTA,TXEN        ; we'll get an almost immediate
interrupt after EI and TXREG will be
                            ; rapidly filled with the first 2 bytes, after
that we can expect interrupts
                            ; every ~100uS. Don't try to put anything in the
TX buffer. If you do, a
                            ; wait up to TXBUFFZIZE*100uS (while it empties)
will occur
        MOVFF        TXTEMPSTATB,STATUS    ; restore the bank bits
        EI
        RETURN

; queue a character in the next free space in the buffer. If the buffer
fills
; then it will call txstart to empty the buffer to make room.
; routine must be single threaded. If you write it from the ISR, chance it
happens
; while you were writing it anyway, regs get corrupted and it crashes the
system
TXBUFFQ:
        MOVWF        TXCHTEMP        ; save the char
        MOVFF        STATUS,TXTEMPSTAT    ; preserve the bank bits
        MOVFF        FSR,TXTEMPFSR
        BTFSC        TXINPROGRESS            ; if we are emptying the
buffer, we must wait before we can proceed
        GOTO        $-1

        DI                    ; other things use FSR
        BANK2                    ; all TX Buffers & ptrs are in BANK2
        MOVLF        LOW TXBUFF,FSR            ; point to the start of the
buffer...
        MOVFW        TXBUFFPTR        ; add the pointer
        ADDWF        FSR            ; here INDF is the nth character in the
buffer
        MOVFF        TXCHTEMP,INDF            ; put the char in the buffer
        INCF        TXBUFFPTR        ; move thge pointer along
        INCF        FSR            ; point to the next position
        CLRF        INDF            ; always write a zero byte after each
char. automatically inserts EOB char
        CP        TXBUFFPTR,TXBUFFSIZE-1   ; check for end of buffer
        CALLZ        TXSTART            ; transmit buffer is full so empty
it
```

```
        MOVFF        TXTEMPFSR,FSR
        MOVFF        TXTEMPSTAT,STATUS    ; restore the bank bits

        EI                    ; interrupts potentially been delayed 30-ish
uS but it is tidy this way
        RETURN



;reset the RX BUFFER
RXBUFFCLEAR:
        DI
        MOVFF        STATUS,RXCHTEMP            ; preserve the bank bits
        BANK1
        CLRF        RXBUFFRDPTR
        CLRF        RXBUFFWRPTR
        LO        RXBUFFERFULL
        LO        RXBUFFEROVF
        LO        RXBUFFACTIVE
        LO        FLAGS,4
        MOVFF        RXCHTEMP,STATUS            ; restore the bank bits
        EI
        RETURN



; shuffle the top of the buffer down. from RXBUFFERPTR to zero
; this way we can recieve partial bits and still leave them in a
; state they can be parsed sequentially, i.e. we don't have to
; take everything in the buffer in one go
RXBUFFSHUFFLE:
        DI
        MOVFF        STATUS,TXTEMPSTAT    ; preserve the bank bits - using
TX temp stat !
        BANK1

BUFFEMPTY:                      ; this is  exit for the read routine; if
there was nothing to
                            ; read, either coz the buffer is empty or coz
the WR & RD pointers
                            ; are the same, we try to do a shuffle to keep
things tidy
        MOVFW        RXBUFFWRPTR
        JMPZ        NOSHUFFLE        ; if WR is already zero, then nothing
to do

;adjust the WR pointer
        MOVFW        RXBUFFRDPTR        ; WR pointer - RD pointer = new WR
pointer
        JMPZ        NOSHUFFLE        ; if RD is zero, we have no where to go

        SUBWF        RXBUFFWRPTR        ; otherwise compute a new place to
write to
```

```
; calculate the source & destination pointers in the buffer
        MOVLF        LOW RXBUFF,RXSHUFFDST    ; destination for the data
        ADDWF        RXBUFFRDPTR,W
        MOVWF        RXSHUFFSRC        ; source of the data
;now go round in a loop until the pointer is at the end of the buffer+1
(after the INCF)
SHUFFLOOP:
        MOVFF        RXSHUFFSRC,FSR        ;move the byte
        MOVFF        INDF,RXCHTEMP
        MOVFF        RXSHUFFDST,FSR
        MOVFF        RXCHTEMP,INDF

;calculate new positions
        INCF        RXSHUFFSRC
        INCF        RXSHUFFDST
        INCF        RXBUFFRDPTR
        CP        RXBUFFRDPTR,RXBUFFSIZE+1; have we reached the buffer end
        JMPNZ        SHUFFLOOP        ; go again if not

        MOVFW        RXBUFFWRPTR        ; otherwise, point to first position
(where our data
        SKIPNZ
        LO        RXBUFFACTIVE        ; if the WR pointer is 0 then the
buffer is empty
        LO        RXBUFFERFULL        ; we shuffled so the buffer can't be
full

NOSHUFFLE:
        CLRF        RXBUFFRDPTR
        MOVFF        TXTEMPSTAT,STATUS    ; restore the bank bits
        MOVLW        0            ; this is here for the read exit
        EI
        RETLW        0

; read a character from the RXBUFFER
RXBUFFREAD:
        DI
        MOVFF        STATUS,TXTEMPSTAT    ; preserve the bank bits - using
TX temp stat !
        BANK1
        MOVFW        RXBUFFWRPTR        ; if the write pointer is zero,
nothing there
        JMPZ        BUFFEMPTY

        SUBWF        RXBUFFRDPTR,W        ; compare RD & WR pointers, don't
care so long as they not the same
        JMPZ        BUFFEMPTY        ; if they are attempt a shuffle

        ;looks good, lets compute the buffer position and get our character
        MOVLW        LOW RXBUFF        ; destination for the data
        ADDWF        RXBUFFRDPTR,W
        MOVWF        FSR
        MOVFF        INDF,DATAL
        INCF        RXBUFFRDPTR        ; move the read ptr along
        MOVFF        TXTEMPSTAT,STATUS    ; restore the bank bits
        MOVFW        DATAL            ; in W
        EI
        RETURN


; end RS232 module
```