A software duplex UART for the 751/752 AN446

Author: Greg Goodhue

The following program contains routines that will allow an 8xC751 or 8xC752 to implement a software UART that can send and receive serial data simultaneously. Other published software UARTs only allow either transmit or receive to occur at any one time. The demo application shown in the code listing waits for data to be received, then echoes it and follows this with a hexadecimal interpretation of the data plus a space. For instance, if the program receives the character "$", it echoes back the string "$24". The reason for echoing these additional characters is to make it easy to force the receiver buffer to fill up in order to test the handshaking. If the program simply echoed what was received, it would likely never use more than the very first receiver buffer location since it can normally transmit just as fast as it can receive.

CHIP RESOURCES

The UART routines use about 400 bytes of code space and use the timer to provide a constant time interrupt to synchronize both transmit and receive operations. The hardware connections require four device pins to accomplish serial I/O with RTS/CTS handshaking. Only two pins would be needed if handshaking is not required. Three of the four pin functions may be assigned to any port pin. The serial input pin must be assigned as one of the external interrupt pins. Another two pins are used in the demo application to input a selection of one of four baud rates (1200, 2400, 4800, or 9600).

LIMITATIONS

To obtain duplex operation, a fairly large portion of the chip's time is used. The routines were tested up to 9600 baud running on a 16 MHz 87C751. When serial input and output were both occurring at the same time, the routines could not support continuous operation with no pauses between characters. At 4800 baud, full speed tight reception and transmission worked flawlessly. In other words, 4800 baud should work with all applications, while 9600 baud may not work with all applications.

THEORY OF OPERATION

There are three possible sequences of events when serial transmit and receive may both be operating at once: transmit and receive begin simultaneously; transmit is requested while the receiver is busy; and receive starts while the transmitter is busy. The first 2 cases could be handled fairly simply with only one interrupt for each bit time. In the first case, everything is already in synch and only one timer and one interrupt per bit is needed to do both operations. In the second case (transmit is requested while the receiver is busy) the program could just wait for the next bit time to start transmitting. Unfortunately, the third case presents a problem. If the program is already transmitting, it cannot always wait for the next bit time to start sampling the serial data if the application is not to lose bits. Also, the timer cannot be adjusted to the incoming data since this would distort the duration of one of the transmitted bits.

The method used here to deal with this problem is to always divide all bit times into 4 sub-bit times. When transmission and/or reception is in progress, the timer runs at 4X the bit rate for the selected baud rate. The variables TxTime and RxTime are used to count sub-bit times for the transmitter and the receiver, respectively. Both are initialized to a negative value and count up to simplify testing for an active sub-bit time. The maximum baud rate that can be supported is essentially determined by the maximum amount of time that it might take the microcontroller to do all of the operations associated with transmitting one bit and receiving one bit . This must be done within the time between timer interrupts.

When both transmit and receive operations are scheduled for the same timer interrupt, priority is given to the transmitter routine. The reason for this is that a great deal of jitter can be tolerated in the timing of the received bit sampling, but the transmitted data must "look" good to the outside world.

The actual bit times for transmit and receive are counted by the variables TxCnt and RxCnt, respectively. When an active sub-bit time slice occurs, these variables tell the transmit and receive routines what to do in the current time slice. The value 11 hex indicates a start bit, 10 hex indicates a stop bit, and the values 8 through F hex indicate a data bit. The values were chosen to allow quick determination of the appropriate action by the code.

The routines provide for a small amount of data buffering for both the transmitter and the receiver. As implemented here, the transmitter buffer is only one byte deep, allowing one data byte to be held while another is being transmitted. The receiver buffer is larger, allowing three bytes to be held while a fourth is being received. If the receiver buffer fills up (indicated by the flag RxFull), the application code must retrieve one byte before a fourth one finishes, or data will be lost. If this happens, a flag will be set (OverrunErr) to indicate that the receiver buffer has been overrun. There is no similar flag for the transmitter, since the transmit request routine waits for the transmitter buffer to be available (indicated by the TxFull flag) before taking action. It is up to the application code to check this flag in advance if it does not want to stall execution while waiting to transmit data.

As each routine finishes a whole data byte by completing the send or receive of a stop bit, it checks to see if there is something still happening to warrant having the time slice interrupt running. In the case of a received stop, the transmit activity flag (TxOn) is examined. If it is not set, the timer is turned off. The timer will be turned back on if an interrupt from a serial start bit is received or the main code requests data to be transmitted. In the case of a transmitted stop, both the receiver activity flag (RxOn) and the transmit buffer flag (TxFull) are examined. If the receiver is active or there is more data to transmit, the timer is left running.

All of the status flags are in the "Flags" register. Other status flags found there are: RxAvail, which indicates that the receiver buffer contains unprocessed data; and FramingErr which is set when the receiver routines find an improper start or stop bit, usually caused by mismatched baud rates.

Flow control handshaking is provided by the RTS/CTS scheme. The transmit routine looks at the incoming CTS line before beginning each start bit transmission, and simply exits, waiting for the next time slice, if CTS is not asserted. The receive routine checks the buffer status whenever a start bit interrupt occurs and de-asserts the outgoing RTS line if the buffer already contains two bytes (i.e., it will be full when the current byte finishes). If the device at the other end of the communication line follows the same rules (which may very well NOT be the case) the program should be able to communicate without buffer overflows in either direction.

Baud rates in both the send and receive routines are determined by two things: the timer interrupt rate; and the number of time slices per bit. The method of calculating the timer value for various baud rates is discussed in the code listing at the BaudRate routine. This discussion has centered on there being four time slices per bit, but if the user wants, either the transmitter or the receiver can be set to run at a baud rate that is a multiple of the other by adjusting the value of the constant TxBitLen or RxBitLen. The baud rate would be calculated as indicated for the faster channel, and TxBitLen or RxBitLen would be changed for the slower channel. For example, the transmitter can be set to run at half of the receiver baud rate by setting TxBitLen to –8 + 1.

The routines shown also make provision for changing the baud rate “on the fly”, although the application code given does not implement this feature. If the application code changes the baud rate for some reason, the change will be effected when the next data transmission or reception begins, if both the transmitter and receiver were already idle. This prevents the timer value from being changed in the middle of a data byte.

THE CODE

There are a number routines in the code of which the user should be aware:

- Intr0—Called (by interrupt) when a serial start bit is received.
- Timer0—Called (by interrupt) for every sub-bit time slice.
- RS232TX—Called by Timer0 when the transmitter has business to conduct in the current time slice.
- RS232RX—Called by Timer0 when the receiver has business to conduct in the current time slice.
- BaudRate—Sets the baud rate variables BaudHigh and BaudLow based on the accumulator value.
- TxSend—Called by the application code to request that a data byte be transmitted. The data to be transmitted is in the accumulator.
- GetRx—Called by the application code to request return of a received data byte from the buffer. Data is returned in the accumulator. This routine should not be called unless the receiver buffer has data available.
- Reset—Start of the initialization code to set up the UART.
- MainLoop—Start of the mainline code of the demo application.

```
;*****************************************************************************
;        Duplex UART Routines for the 8xC751 and 8xC752 Microcontrollers
;*****************************************************************************

; This is a demo program showing a way to perform simultaneous RS–232
; transmit and receive using only one hardware timer.

; The transmit and receive routines divide each bit time into 4 slices to
; allow synchronizing to incoming data that may be out of synch with outgoing
; data.

; The main program loop in this demo processes received data and sends it
; back to the transmitter in hexadecimal format. This insures that we can
; always fill up the receiver buffer (since the returned data is longer than
; the received data) for testing purposes. Example: if the letter "A" is
; received, we will echo "A41 ".
;*****************************************************************************

$Title(Duplex UART Routines for the 751/752)
$Date(8/20/92)
$MOD751

;*****************************************************************************
;                                  Definitions
;*****************************************************************************

; Miscellaneous

TxBitLen    EQU   –4 + 1               ; Timer slices per serial bit transmit.
RxBitLen    EQU   –4 + 1               ; Timer slices per serial bit receive.
RxHalfBit   EQU   (RxBitLen / 4) + 1   ; Timer slices for a partial bit time.
                                       ;   Used to adjust the input sampling
                                       ;   time point.

; Note:  TxBitLen and RxBitLen are kept separate in order to facilitate the
;   possibility of having different transmit and receive baud rates. The timer
;   would be set up to give four slices for the fastest baud rate, and the
;   BitLen for the slower channel would be set longer for the slower baud rate.
;   BitLen = –4 + 1 gives four timer interrupts per bit. BitLen = –8 + 1 would
;   give 8 slices, BitLen = –16 + 1 would give 16 slices, etc.

TxPin       BIT   P1.0                 ; RS–232 transmit pin (output).
RxPin       BIT   P1.5                 ; RS–232 receive pin (input).
RTS         BIT   P1.3                 ; RS–232 request to send pin (output).
CTS         BIT   P1.6                 ; RS–232 clear to send pin (input).
; Note: P1.1 and P1.2 are used to input the baud rate selection.

; RAM Locations

Flags       DATA  20h                  ; Miscellaneous bit flags (see below).
TxOn        BIT   Flags.0              ; Indicates transmitter is on (busy).
RxOn        BIT   Flags.1              ; Indicates receiver is on (busy).
TxFull      BIT   Flags.2              ; Transmit buffer (1 byte only) is full.
RxFull      BIT   Flags.3              ; Receiver buffer is full.
RxAvail     BIT   Flags.4              ; RX buffer is not empty.
OverrunErr  BIT   Flags.6              ; Overrun error flag.
FramingErr  BIT   Flags.7              ; Framing error flag.

BaudHigh    DATA  21h                  ; High byte timer value for baud rate.
BaudLow     DATA  22h                  ; Low byte timer value for baud rate.

TxCnt       DATA  23h                  ; RS–232 byte transmit bit counter.
TxTime      DATA  24h                  ; RS–232 transmit time slice count.
TxShift     DATA  25h                  ; Transmitter shift register.
TxDat       DATA  26h                  ; Transmitter holding register.
```

```
RxCnt       DATA  27h                   ; RS-232 byte receive bit counter.
RxTime      DATA  28h                   ; RS-232 receive time slice count.
RxShift     DATA  29h                   ; Receiver shift register.
RxDatCnt    DATA  2Ah                   ; Received byte count.
RxBuf       DATA  2Bh                   ; Receive buffer (3 bytes long).

Temp        DATA  2Fh                   ; Temporary holding register.

;*****************************************************************************
;                               Interrupt Vectors
;*****************************************************************************

            ORG   00h                   ; Reset vector.
            AJMP  RESET

            ORG   03h                   ; External interrupt 0
            AJMP  Intr0                 ; (received RS-232 start bit).

            ORG   0Bh                   ; Timer 0 overflow interrupt.
            AJMP  Timer0                ; (4X the RS-232 bit rate).

            ORG   13h                   ; External interrupt 1.
            RETI                        ; (not used).

            ORG   1Bh                   ; Timer I interrupt.
            RETI                        ; (not used).

            ORG   23h                   ; I2C interrupt.
            RETI                        ; (not used).

;*****************************************************************************
;                              Interrupt Handlers
;*****************************************************************************

; External Interrupt Int0.
;   RS-232 start bit transition.

Intr0:      PUSH  ACC                   ; Save accumulator,
            PUSH  PSW                   ;   and status.
            CLR   IE.0                  ; Disable more RX interrupts.

            SETB  RxOn                  ; Set receive active flag.
            MOV   RxCnt,#11h            ; Set bit counter to expect a start.
            MOV   RxTime,#RxHalfBit     ; First sample is at a partial bit time.
            JB    TxOn,I0TimerOn        ; If TX active then timer is on.

            MOV   RTH,BaudHigh          ; Set up timer for selected baud rate.
            MOV   RTL,BaudLow
            MOV   TH,BaudHigh
            MOV   TL,BaudLow
            SETB  TR                    ; Start timer 0.

I0TimerOn:  MOV   A,RxDatCnt            ; Check for buffer about to be full:
            CJNE  A,#2,Int0Ex           ;   one space left and a byte starting.
            SETB  RTS                   ; If so, tell whoever is on the
                                        ;   other end to wait.

Int0Ex:     POP   PSW                   ; Restore status,
            POP   ACC                   ;   and accumulator.
            RETI

; Timer 0 Interrupt
;   This is used to generate time slices for both serial transmit and receive
;   functions.
```

```
Timer0:     PUSH  ACC                 ; Save accumulator,
            PUSH  PSW                 ;   and status.
            JNB   TxTime.7,RS232TX    ; Is this an active time slice
                                      ;   for an RS–232 transmit?
            JNB   TxOn,CheckRx        ; If transmit is active,
            INC   TxTime              ;   increment the time slice count.
CheckRx:    JNB   RxTime.7,RS232RX    ; Is this an active time slice
                                      ;   for an RS–232 receive?
            JNB   RxOn,T0Ex           ; If receive is active, increment
            INC   RxTime              ;   the time slice count.

T0Ex:       POP   PSW                 ; Restore status,
            POP   ACC                 ;   and accumulator.
            MOV   P3,Flags            ; For demo purposes, output status
                                      ;   on an extra port.

            RETI

;*****************************************************************************
;                            RS–232 Transmit Routine
;*****************************************************************************

RS232TX:    JNB   TxCnt.4,TxData      ; Go if data bit.
            JNB   TxCnt.0,TxStop      ; Go if stop bit.

; Send start bit and do buffer housekeeping.

TxStart:    JB    CTS,TxEx1           ; Is CTS asserted (low) so can we send?
                                      ;   If not, try again after 1 bit time.
            CLR   TxPin               ; Set start bit.
            MOV   TxShift,TxDat       ; Get byte to transmit  from buffer.
            CLR   TxFull
            MOV   TxCnt,#08h          ; Init bit count for 8 bits of data.
                                      ;   (note: counts UP).

TxEx1:      MOV   TxTime,#TxBitLen    ; Reset time slice count.
            SJMP  CheckRx             ; Restore state and exit.

; Send Next Data Bit.

TxData:     MOV   A,TxShift           ; Get un–transmitted bits.
            RRC   A                   ; Shift next TX bit to carry.
            MOV   TxPin,C             ; Move carry out to the TXD pin.
            MOV   TxShift,A           ; Save bits still to be TX'd.
            INC   TxCnt               ; Increment TX bit counter
            MOV   TxTime,#TxBitLen    ; Reset time slice count.
            SJMP  CheckRx             ; Restore state and exit.

; Send Stop Bit and Check for More to Send.

TxStop:     SETB  TxPin               ; Send stop bit.
            JB    TxFull,TxEx2        ; More data to transmit?
            CLR   TxOn                ; If not, turn off TX active flag, and
            CLR   RTS                 ;   make sure that whoever is on the
                                      ;   other end knows it's OK to send.

            JB    RxOn,TxEx2          ; If receive active, timer stays on,
            CLR   TR                  ;   otherwise turn off timer.

TxEx2:      MOV   TxCnt,#11h          ; Set TX bit counter for a start.
            MOV   TxTime,#TxBitLen–1  ; Reset time slice count, stop bit
                                      ;   > 1 bit time for synch.
            SJMP  CheckRx             ; Restore state and exit.
```

```
;*******************************************************************************
;                              RS-232 Receive Routine
;*******************************************************************************

RS232RX:    MOV   C,RxPin               ; Get current serial bit value.
            JNB   RxCnt.4,RxData        ; Go if data bit.
            JNB   RxCnt.0,RxStop        ; Go if stop bit.

;Verify start bit.

RxStart:    JC    RxErr                 ; If bit=1, then not a valid start.
            MOV   RxCnt,#08h            ; Init counter to expect data.
            MOV   RxTime,#RxBitLen      ; Reset time slice count.
            SJMP  T0Ex                  ; Restore state and exit.

; Get Next Data Bit.

RxData:     MOV   A,RxShift             ; Get partial received byte.
            RRC   A                     ; Shift in new received bit.
            MOV   RxShift,A             ; Store partial result in buffer.
            INC   RxCnt                 ; Increment received bit count.
            MOV   RxTime,#RxBitLen      ; Reset time slice count.
            SJMP  T0Ex                  ; Restore state and exit.

; Store Data Byte, "push"ing it into the FIFO buffer.

RxStop:     CLR   EA                    ; Don't interrupt the following.
            MOV   A,RxBuf               ; "PUSH" the receive buffer.
            XCH   A,RxBuf+1
            XCH   A,RxBuf+2
            MOV   RxBuf,RxShift         ; Add just completed data to buffer.
            INC   RxDatCnt              ; Increment the received byte count.
            SETB  EA                    ; Re-enable interrupts.

            SETB  RxAvail               ; There is data in the RX buffer.
            PUSH  PSW                   ; Save Carry (received bit)for later.
            MOV   A,RxDatCnt            ; Check receiver buffer status.
            CJNE  A,#4,RxChk1           ; Is RX buffer overrun?
            SETB  OverrunErr            ; Set status reg overrun error flag.
            MOV   RxDatCnt,#3           ; Re-set buffer counter to "full".

RxChk1:     CJNE  A,#3,RxChk2           ; Is RX buffer full?
            SETB  RxFull                ; Set buffer full status.

RxChk2:     POP   PSW                   ; Retrieve last received bit in Carry.
            JC    RxEx                  ; If bit=0, then not a valid stop.
RxErr:      SETB  FramingErr            ; Remember bad start or stop status.

RxEx:       JB    TxOn,RxTimerOn        ; If transmit active, timer stays on,
            CLR   TR                    ;   otherwise turn timer off.
RxTimerOn:  CLR   RxOn                  ; Turn off receive active.
            SETB  RxTime.7              ; Set bit for no service to
                                        ;   RX Time Slice Branches.
            SETB  IE.0                  ; Re-enable RS-232 receive interrupts.
            AJMP  T0Ex                  ; Restore state and exit.


;*******************************************************************************
;                                  Subroutines
;*******************************************************************************

; BaudRate - Determine and set the baud rate from switches.
;   Note: if the baud rate is altered, the actual change will only occur when
;   a transmit or receive is begun while the timer was not already running
;   (i.e.: not already busy transmitting or receiving).
```

```
BaudRate:   MOV   DPTR,#BaudTable       ; Set pointer to baud rate table.
            ANL   A,#03h                ; Limit displacement for lookup.
            RL    A                     ; Double the table index since these
                                        ;   are 2 byte entries.
            PUSH  ACC                   ; Save the table index for second byte.
            MOVC  A,@A+DPTR             ; Get first byte, and save as the high
            MOV   BaudHigh,A            ;    byte of the baud rate timer value.
            POP   ACC                   ; Get back the table index.
            INC   A                     ; Advance to next table entry.
            MOVC  A,@A+DPTR             ; Get second byte, and save as the low
            MOV   BaudLow,A             ;    byte of the baud rate timer value.
            RET

; Entries in BaudTable are for a timer setting of 1/4 of a bit time at the given
;   baud rate. The two values per entry are the high and low bytes of the value
;   respectively.

; Values are calculated as follows:
;                                              Osc Frequency
;   1/4 Bit cell time (in machine cycles) = ------------------
;                                           Baud Rate * 48

; Example for 9600 baud with a 16MHz crystal:
;    16,000,000 / 9600 * 48 = 34.7222... machine cycles per quarter bit time.
;    Rounded, this is 35. The hexadecimal value for 35 is 23.
;    10000 hex – 23 hex (truncated to 16 bits) = FFDD. Thus, the BaudTable entry
;    for 9600 baud is FF, DD hex.

BaudTable:  DB    0FEh,0EAh             ; 1200 baud.
            DB    0FFh,75h              ; 2400 baud.
            DB    0FFh,0BBh             ; 4800 baud.
            DB    0FFh,0DDh             ; 9600 baud.
; TxSend – Initiate RS–232 Transmit.

TxSend:     JB    TxFull,$              ; Make sure TX buffer is free.
            SETB  TxFull                ; Reserve the buffer for our use.
            MOV   TxDat,A               ; Put character in buffer.
            JB    TxOn,TSTimerOn        ; Exit if transmitter already running.

            SETB  TxOn                  ; Transmit active flag set.
            MOV   TxCnt,#11h            ; Init bit counter to expect a start.
            MOV   TxTime,#TxBitLen      ; Reset time slice count.
            JB    RxOn,TSTimerOn        ; Exit if receiver already active.

            MOV   RTH,BaudHigh          ; Set up timer for selected baud rate.
            MOV   RTL,BaudLow
            MOV   TH,BaudHigh
            MOV   TL,BaudLow
            SETB  TR                    ; Start up the bit timer.
TSTimerOn:  RET

; PrByte – Output a byte as ASCII hexadecimal format.

PrByte:     PUSH  ACC                   ; Print ACC contents as ASCII hex.
            SWAP  A
            ACALL  HexAsc               ; Print upper nibble.
            ACALL  TxSend
            POP   ACC
            ACALL  HexAsc               ; Print lower nibble.
            ACALL  TxSend
            RET

; HexAsc – Convert a hexadecimal nibble to its ASCII character equivalent.

HexAsc:     ANL   A,#0Fh                ; Make sure we're working with only
                                        ;   one nibble.
            CJNE  A,#0Ah,HA1            ; Test value range.
HA1:        JC    HAVal09               ; Value is 0 to 9.
```

```
            ADD   A,#7                  ; Value is A to F, needs pre-adjustment.
HAVal09:    ADD   A,#'0'                ; Adjust value to ASCII hex.
            RET

; GetRx - Retrieve a byte from the receive buffer, and return it in A.

GetRx:      CLR   EA                    ; Make sure this isn't interrupted.
            DEC   RxDatCnt              ; Decrement the buffer count.
            MOV   A,RxDatCnt            ; Get buffer count.
            JNZ   GRX1                  ; Test for empty receive buffer.
            CLR   RxAvail               ; If empty, clear data available status.
GRX1:       ADD   A,#RxBuf              ; Create a pointer to end of buffer.
            MOV   Temp,R0               ; Save R0.
            MOV   R0,A                  ; Put pointer where we can indirect.
            MOV   A,@R0                 ; Get last buffer data.
            MOV   R0,Temp               ; Restore R0.
            CLR   RxFull                ; Buffer can't be full anymore.
            SETB  EA                    ; Re-enable interrupts.
            RET


;*****************************************************************************
;                                    Reset
;*****************************************************************************
Reset:      MOV   SP,#2Fh               ; Initialize stack start.
            MOV   TCON,#0               ; Set timer off, INT0 to level trigger.
            MOV   P3,#0                 ; Turn off all status outputs.

; For this demo, we only set up the baud rate once at reset:

            MOV   A, P1                 ; Read baudrate bits from P1.
            RR    A                     ; The switches are on bits 2 and 1.
            ACALL BaudRate              ; Set up the selected baud rate.

            MOV   FLAGS,#0              ; Init all status flags.
            MOV   RxDatCnt,#0           ; Clear buffer count.
            MOV   IE,#93h               ; Turn on timer 0 interrupt and
                                        ;   external interrupt 0.
            CLR   RTS                   ; Assert RTS so we can receive.

; The main program loop processes received data and sends it back to the
;   transmitter in hexadecimal format. This insures that we can always fill
;   up the receiver buffer (since the returned data is longer than the
;   received data) for testing purposes. Example: if the letter "A" is
;   received, we will echo "A41 ".

MainLoop:   JNB   RxAvail,$             ; Make sure an input byte is available.
            ACALL GetRx                 ; Get data from the receiver buffer.
            ACALL TxSend                ; Echo original character.
            ACALL PrByte                ; Output the char in hexadecimal format,
            MOV   A,#20h                ;   followed by a space.
            ACALL TxSend
            SJMP  MainLoop              ; Repeat.

            END
```