BREAK DETECTION ON Z80180 AND Z181

Break sequence is a key area not automatically dealt with by all devices however, it is possible to handle break sequences through simple edge detection and hardware configuration techniques.

INTRODUCTION

Within the Z80® product family, there is a group of products based on the Z180™ MPU Core. They are the Z180 (Z80180, Z8S180), Z181™ and Z182. The Z180 MPU is a Superintegration™ device built around a Z80 compatible CPU core, and also includes two UART channels, two DMA channels, two 16-bit timers, CSIO, and other system functions. The on-board UART (universal asynchronous receiver/transmitter) channels meet most common asynchronous communication requirements except break sequence handling. These UART channels lack the embedded hardware features necessary to send/detect break. This application note describes how to handle break.

WHAT IS THE BREAK SEQUENCE?

In the asynchronous communication environment, the "break sequence" is commonly used to interrupt the process. The sequence is defined as the contiguous transmission of "0" (space) for a certain length of time. The CCITT "blue book" specification states that the time duration for this is larger than 2M+3 bit time (where M is the character length). After the break sequence, another 2M+3 bit time consisting of the contiguous transmission of "1" (mark) is required to start the next character. Most serial communication devices, including Z8 x 30 SCC and Z844x SIO, define this condition as "null character with framing error". These devices usually generate interrupt at both the start and the end of the break sequence.

Z180'S BEHAVIOR RELATIVE TO THE BREAK SEQUENCE

When the Z180 UART channel sees the break sequence, it behaves as follows:

If the character bit length is "M", without parity; numbers are bit cell time at given transmission speed; and T is the duration of the space condition; then :

T < 0.5	Receive nothing
0.5 < T < 1.5	Receive all "1" data without error
1.5 < T < 2.5	Receive all "1" data, except D0 location receives "0" without error.
2.5 < T < 3.5	Receive all "1" data, except D1, D0 locations receive "0" without error.
M + 0.5 < T < M + 1.5	Receive all "0" data without error. (This is the condition which normally receives "0".)
M + 1.5 < T < M + 2.0	Receive all "0" data with framing error.
M + 2.0 < T < M + 3.0	Receive all "0" data with framing error, followed by all "1" data without error.
M + 3.0 < T < M + 4.0	Receive all "0" data with framing error, followed by all "1" data, except D0 location receives "0" without error.

Z180'S BEHAVIOR RELATIVE TO THE BREAK SEQUENCE (Continued)

M + 4.0 < T < M + 5.0	Receive all "0" data with framing error, followed by all "1" data, except D1, D0 locations receive "0" without error.
M + 8.0 < T < M + 9.0	Receive all "0" data with framing error, followed by all "0" data without error.
M + 9.0 < T < M + 9.5	Receive all "0" data with framing error, followed by all "0" data with framing error.

In other words, Z180 UART receives the all 0 data with framing error, but may receive the trailing end of the character without error. The all 0 character with framing error will be continuously received as long as break condition exists.

IMPLEMENTATION

To send break sequence requires external hardware. One approach, as shown in the following (Figure 1), is to have one gate on the TxD pin and mask off the status when the break sequence is needed. The I/O port to mask TxD could be one of the modem control signals on-chip PIO or PIA. Since the TxD pin is forced to 0 while the control signal is active, writing a dummy character and using Tx interrupt is one possible way to time the duration.

Knowing the Z180's behavior also makes it possible to detect break sequence as follows. On the reception of the all 0 character with framing error, disable receiver and discard the character, and re-enable the receiver on the rising edge of the receive data line (the end of break sequence). For this purpose, the Z80 CTC can be used to detect the edge. On the Z181, the on-chip Z80 CTC can be used.

Z180's interrupts from the on-chip UART are handled through vectored interrupt, regardless of the interrupt mode programmed through instruction (IM0/1/2). You can enable the interrupt for transmitter and receiver separately, but the vector for both will be the same. So, the interrupt routine has to handle both interrupts, from the receiver and the transmitter. Upon interrupt, the interrupt handing routine has to poll the status register (stat0), judge the cause and initiate a correction.

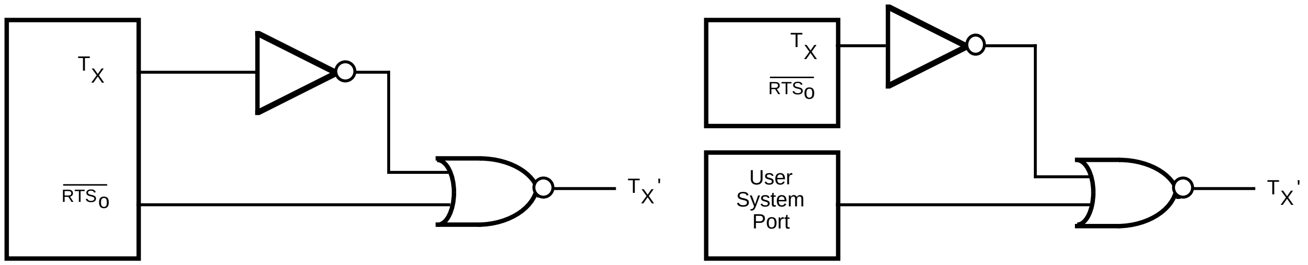

Figure 1. Break Sequence Generation

PROGRAMMING EXAMPLE

The following program example was written for the Z181™ using Zilog's Z181 evaluation board (P/N: Z8018100ZCO), and verified to work up to 38400 Kbps. As discussed above, this program is interrupt-driven, and uses on-chip CTC channel 0 and 1 for edge detection. One channel is used for rising edge detection, and the other channel is used to detect the falling edge of the data. On the interrupt from ASCI0, this program reads the stat0 and checks for errors. If there was a framing error, then read out the character and check if it is zero. If that character was 00, then disable the receiver and enable CTC0/CTC1 for edge detection. When it detects edge, re-enable the receiver. Enabling interrupts for both edges accounts for cases when the rising edge of the data comes before enabling CTC (for rising edge detection). In these cases, the falling edge which needs to be detected is the falling edge of the start bit of the next character.

This program has one shortcoming. If the pulse width is too short to recognize the following character (as all "1" data), the rising edge of the data may appear before the CTC is enabled for rising edge detection, depending on the processing speed/overhead, In this case, enabling the receiver after the start bit will throw off the sampling point of the data slightly. The extent of the deviation will vary depending on the interrupt response time for the CTC interrupt.

CONCLUSION

As discussed, the Z180's UARTs do not have an automatic break detection circuit; however, it is possible to handle the break sequence using edge detection, and with additional hardware one can also generate break.

```
;***********************************  *
;*                                    *
;*   Break detection for Z181         *
;*        April 5, 1991               *
;*       By Jim Nobugaki              *
;*                                    *
;***********************************

; This is a break detection program for ASCI0.
;
; The environment assumed is the Z181 application board
;(Z8018100ZCO), and
; the clock speed is 12.288 MHz.
;
; It is an interrupt-driven program using CTC0 for rising
; edge detect, CTC1 for falling edge detect
;

.z800

*include 181macro.lib               ;Read in Z180 register
                                    ;names and
                                    ;macro for Z180 new
                                    ;instructions

         org      08400h ;          top of RAM physical
                                    ;address

bkdet:   ld       sp,stack          ;set up the stack pointer

         ld       a,high vecttab    ;init i reg
         ld       i,a

         ld       a,00h             ;init il
         out0     (il),a

         im       2                 ;set int mode 2

         ld       a,low ctcvect
         out0     (ctc0),a          ;init CTC int vect

         ld       hl,bkend
         ld       (ctcvect),hl

         ld       hl,bkend
         ld       (ctcvect+2),hl

         ld       hl,asci0int       ;set asci0 int vect
         ld       (vecttab+0eh),hl

         ld       a,01000001b       ;configure pia1 as ctc i/o
         out0     (scr),a

         ld       a,00010001b       ;set 38400 bps
         out0     (cntlb0),a        ;change this for desired
                                    ;speed

         ld       a,00001000b       ;enable only rx int
         out0     (stat0),a

         xor      a                 ;clear stat flag
         ld       (bip),a

         ld       a,01010000b       ;select cnta0
         out0     (cntla0),a        ;only Rx enable!

         ei
```

CONCLUSION (Continued)

```
wait_here:      jr      wait_here       ;wait here forever

;int service routine for asci0
;

asci0int:       push    af              ;save regs to be used
        push    bc

        in0     a,(stat0)               ;read stat
        ld      b,a                     ;save stat info into b

        and     0f0h                    ;mask info

        jp      z,asci0int1             ;no rec related int...

        and     070h                    ;any errors ?
        jr      nz,rec_err

        ;;;;;;;;
        ;process for normal rec'd char here
        ;;;;;;;;
        in0     a,(rdr0)                ;read char
        ;;;;;;;;

        jp      asci0int1               ;jump to tx check

rec_err: in0    a,(rdr0)                ;read char
        bit     4,b                     ;see if it is FE?
        jr      z,asci0int2             ;if not, OVRN or PE...

        cp      a,00h                   ;see if it's 00h+FE
        jr      z,asci0int3

        ;;;;;;;;
        ;here comes simple FE...
        ;;;;;;;;



        ld      a,01000000b             ;error reset
        out0    (cntla0),a
        jp      asci0int1               ;jump to tx check
asci0int3:      ld      a,00010000b
                                        ;disable Rx,
                                        ;with error reset
        out0    (cntla0),a

        ld      a,11010101b             ;start ctc0 for /edge
                                        ;detect
        out0    (ctc0),a
        ld      a,01h                   ;tc=1
```

```
        out0    (ctc0),a

        ld      a,11000101b     ;start ctc1 for \edge
                                detect
        out0    (ctc1),a
        ld      a,01h           ;tc=1
        out0    (ctc1),a

        ld      a,0ffh
        ld      (bip),a         ;set break in process flag.

        ld      a,00000000b     ;shut off rec int!
        out0    (stat0),a

        jp      asci0int1       ;branch to tx int check

asci0int2:      ;;;;;;;;

        ;process for parity error/overrun error here
        ;;;;;;;;

        ld      a,01000000b     ;error reset
        out0    (cntla0),a

asci0int1:      bit     1,b
        jr      z,asci0int4

        ;;;;;;;;
        ;If there's something to send, place routine here
        ;For this case, don't destroy b register!
        ;;;;;;;;

asci0int4:      bit     2,b     ;/dcd int ?
        jr      z,asci0exit

        ;;;;;;;;
        ;if there's something to do with /dcd, place routine
        here
        ;;;;;;;;

asci0exit:pop   bc      ;restore regs
        pop     af
        ei
        ret

;bkend -
;this routine called when detecting edges of rxd,
;which is the end of break (hopefully...  ^_^!).
;if rising edge (int from ctc0), that is the end of break
;if falling edge (int from ctc1), missed rising edge for the
;end of break and that's start of start bit of next char...
```

```
bkend:  push    af          ;save regs

        ld      a,01001011b     ;s/w reset, stop ctc0/1
        out0    (ctc0),a
        out0    (ctc1),a

        ld      a,01010000b     ;select cnta0
        out0    (cntla0),a      ;enable Rx again!

        ld      a,00001000b     ;enable rx int again
        out0    (stat0),a

        xor     a               ;clear stat flag
        ld      (bip),a

        pop     af              ;restore regs

        ei
        reti
```

```
;Interrupt vector table
;
        org     bkdet+200h

vecttab:ds      20h     ;reserve space for z180 vector

ctcvect:ds      8h      ;int vector table for ctc

bip:    ds      1       ;break in progress flag

dummy:ds        0ffh    ;space for stack
stack:

                end
```

Zilog's products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the customer and Zilog prior to use. Life support devices or systems are those which are intended for surgical implantation into the body, or which sustains life whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user.

Zilog, Inc. 210 East Hacienda Ave.
Campbell, CA 95008-6600
Telephone (408) 370-8000
Telex 910-338-7621
FAX 408 370-8056
Internet: http://www.zilog.com