Transfer data frames over asynchronous RS-232C lines

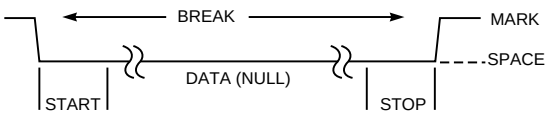
SK SHENOY, NAVAL PHYSICAL AND OCEANOGRAPHIC LAB, KOCHI, INDIA

The asynchronous RS-232C interface is a simple, low-cost option for interconnecting processor-based systems. In many applications, you need to transfer variable-size messages. However, the character-oriented RS-232C protocol offers no direct mechanism for transferring messages as self-contained packets. The method described here uses an obscure feature found in most UART devices to indicate packet boundaries. The feature is the capability to transmit and recognize the "Break" character. This character is nothing but a "space," or low, in the transmit line of a duration equal to or greater than an entire asynchronous character-transmission time, including the start and stop bits (**Figure 1**). In this framing method, the message data bytes sandwich between two Break characters to form a data frame (**Figure 2**).

A Turbo C program demonstrates the transfer of variable-size messages between two PCs with 8250-compatible UARTs (**Listing 1**). You can download the program from *EDN*'s Web site, www.ednmag.com. At the registered-user area, go into the Software Center to download the files from DI-SIG, #2140.

A null-modem cable interconnects the PCs' COM ports. The same routine works with most other UARTs. The method allows data-packet reception in interrupt mode and wastes no CPU overhead looking at each character to detect packet boundaries. Instead, the UART does the detecting. Because the Break is not a legitimate data character, it is data-

FIGURE 1



Most UARTs can transmit and recognize the Break character, a state of logic 0 between the start and stop bits.

FIGURE 2

The Turbo C routine in Listing 1 sandwiches data bytes between two Break characters to form a data frame.

transparent, and you can use it for binary-data exchange. You can use this "in-band" scheme with repeaters and modems, as long as they permit transmission of the Break condition. The packet-boundary detection is relatively immune to a missed Break character and to data errors. You can render the detection more robust by introducing data-length and check-sum fields in the frame to allow detection of errors and flow control using an RTS/CTS (request-to-send/clear-to-send) handshake.

To transmit a Break, set bit 6 (Set Break) of the line-control register to 1. The UART then sets its Tx line low, until bit 6 encounters a 0. Transmission of a Null character (00 hex) makes the duration of the Break equal to one character-transmission delay. Bit 6 of the line-status register (Tx Machine Status) indicates when this delay is over; then, the Break bit resets. To enable detection of the Break, bit 2 of the interrupt-enable register (interrupt-on-Rx-error condition) sets during UART initialization. Bit 0, set to 1, enables receive-data interrupts. In the interrupt-service routine (ISR), bits 1 and 2 of the interrupt-identification register indicate the interrupt type.

A global variable, Receive_Count, initialized to zero, handles frame reception. Upon detection of a Break, the UART raises an interrupt. If Receive_Count is zero, the interrupt is a start-of-frame break and the UART ignores it. (You can use the interrupt to set a Packet_Receive_On flag.) On each subsequent receive interrupt, the ISR stores the data in the Receive buffer with Receive_Count as the index. If Receive_Count is nonzero when the Break interrupt is raised, the interrupt is an end-of-frame break. Then the routine calls the frame-processing function and resets Receive_Count to zero. (DI #2140) EDN

To Vote For This Design, Circle No. 357

LISTING 1—DATA-FRAME-TRANSFER PROGRAM

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

/* COM PORT DEFINITIONS AND GLOBAL VARIABLES */
#define   com_reg   0x3f8 /* Default is com1; 2f8 for com2 */
#define   DATA_PORT  com_reg + 0
#define   LINE_CNTRL com_reg + 3
#define   MODEM_CNTRL com_reg + 4
#define   INT_ENABL  com_reg + 1
#define   INT_IDENT  com_reg + 2
#define   LINE_STS  com_reg + 5
#define   MODEM_STS  com_reg + 6
#define   BAUD_LOW  com_reg + 0
#define   BAUD_HIGH  com_reg + 1
#define   DLAB_SET  0x80
#define   BAUDMSB   0
#define   BAUDLSB   0xc /* 9600 BPS */
#define   CNTRL_CMD  7  /* 8 BIT, 2 STOP BIT, NO PARITY */
#define   WAIT_TX_RDY() while (((inportb(LINE_STS))&0x60)!=0x60)
          /* Check for Tx buf empty & Tx shift reg empty */


unsigned char sdatabuf[256],rdatabuf[256]; /* Send & Recv buffers */
int Receive_Count = 0; /* Counter for data stored in rdatabuf[] */
void interrupt(*OldComHandler)(void);


/* FUNCTION CALLED TO DISPLAY RECEIVED DATA PACKET */
void processdata(void)
{
  int i;
  cprintf("\n\rRX > "); clreol(); /* Received data cursor */
  for (i = 0; i< Receive_Count; i++) /* Display received data */
              putch(rdatabuf[i]);
  cprintf("\n\rTX > "); clreol(); /* Transmitted data cursor */
 }


 /* INTERRUPT SERVICE ROUTINE TO TAKE CARE OF PACKET RECEPTION */
 void interrupt service_sio(void)
 {
  unsigned char iir ;
  iir = (inportb(INT_IDENT) >> 1) & 3; /* Get interrupt type */
  switch(iir)
  {
        case 0: /* Modem status int DSR,CTS,RI,RLSD */
          inportb(MODEM_STS); /* Ignore; reading IIR resets int */
          break;/* reading IIR resets int */
        case 1: /* Tx int */


          break;/* reading IIR resets int */
        case 2: /* Rx int */
          rdatabuf[Receive_Count++] = inportb(DATA_PORT); /* Store packet
          break;
        case 3: /* Rx error ( Break detect etc.) */
          inportb(DATA_PORT);   /* NULL char */
          if(((inportb(LINE_STS))&0x10) == 0x10)
          /* Break detected, Reading LSR Resets int */
            if (Receive_Count) processdata();/* EndOfFrame Break Process
            /* Else Start of Frame Break. Do nothing */
          /* Else Receive error; Drop packet */
          Receive_Count = 0; /* Re-initialise for next packet */
  }
  outportb(0x20,0x20); /* EOI */
  return;
}
```

```
/* FUNCTION TO INITIALISE SERIAL PORT */
void init_serial_io(void)
{
      outp(LINE_CNTRL,DLAB_SET);  /* DLAB_SET */
      outp(BAUD_LOW,BAUDLSB); outp(BAUD_HIGH,BAUDMSB); /* 9600 BAUD */
      outp(LINE_CNTRL,CNTRL_CMD); /* 8 BIT,2 STOP BIT,NO PARITY */
      outp(MODEM_CNTRL,8);     /* DTR,RTS & OUT2 SET */
      OldComHandler = getvect(0xc);/* 0xb for com2 */
      disable();
      setvect(0xc,(service_sio)); /* 0xb for com2 */
      outportb(0x21,((inportb(0x21))&(!0x10)));/* PIC mask word 0x8 for
      outportb(INT_ENABL,0x5); /* IER enable Rx Machine error & RX Data
      enable();
}


/* FUNCTION TO TRANSMIT A BREAK OF ONE CHARACTER DURATION */
void SendBreak(void)
{
   outportb(LINE_CNTRL,inportb(LINE_CNTRL) | 0x40); /* LCR; set break */
   outportb (DATA_PORT,0); /* Send NULL data */
   WAIT_TX_RDY(); /* Wait on TxShift Reg Empty; Null char is shifted out
   outportb(LINE_CNTRL,inportb(LINE_CNTRL) & 0xbf); /* LCR; remove break
}

/* FUNCTION TO TRANSMIT A DATA PACKET */
void SendBuffer(unsigned char packet[], int DatLen)
{
  int i;
  SendBreak(); /* Send START OF PACKET break */
  for (i=0; i<DatLen; i++) /* For each message byte*/
  {
    WAIT_TX_RDY(); /* Wait for Tx Ready */

    outportb (DATA_PORT,packet[i]); /* send one data char */
  }
  WAIT_TX_RDY(); /* Wait on TxShift Reg Empty; last char is shifted out
  SendBreak(); /* Send END OF PACKET break char */
}

/* BARE-BONES APPLICATION; TAKES STRING INPUT (TERMINATED BY ENTER) FROM
KEYBOARD AND TRANSMITS AS A PACKET. ALSO DISPLAYS RECEIVED PACKETS */
void main(void)
{
  int c, count = 0;

  init_serial_io(); /* Initialise serial port */
  printf("\n\rTX > "); /* Transmit Prompt */
  while(1) /* Forever Loop */
  {
      if((c=getche()) == 27) break; /* Exit if Escape key pressed */
      sdatabuf[count++] = c;
      if(c == '\r') /* If Enter Key pressed */
      {
        putch('\n'); clreol(); /* Go to newline */
        cprintf("TX > "); /* Transmit Prompt */
        SendBuffer(sdatabuf,count); /* Transmit Data Packet */
        count = 0; /* Reset Tx data count */
      }
  }
  setvect(0xc,(OldComHandler)); /* Restore int vector; 0xb for com2 */
  outportb(0x21,((inportb(0x21))|(0x10)));/* PIC mask word 0x8 for com2
}
```