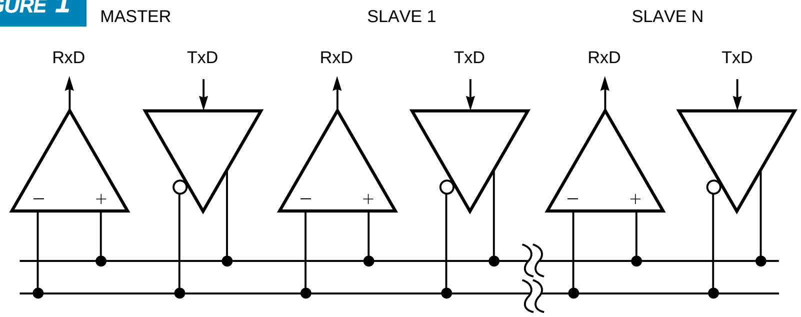
MCS-51 endows MicroLan-like protocol to UARTs

SK Shenoy, NPOL, Kochi, India

µCs such as the 8051 and 8096 and UARTs such as the 82510 provide hardware support for a multiprocessor asynchronous serial-communication protocol (MicroLan). This feature is useful in applications in which a number of processors interconnected in a multipoint configuration jointly perform a task, with a master processor controlling slaves by sending data or commands in a selective manner (**Figure 1**). The protocol operates as follows:

When the master wishes to transmit a block of data to a slave, it first sends an address byte that identifies the slave. All data and address bytes are nine bits long. An address byte differs from a data byte in that its ninth bit is one (for a data byte it's zero). The communication subsystem normally initializes in a mode where the serial-port interrupt activates only when the ninth bit is one. Thus, no slave receives an interrupt from a data byte. An address byte, however, interrupts all slaves, which then examine the received byte. Next, the addressed slave switches to a mode in which data bytes

FIGURE 1



A master-slave arrangement uses RS-422 transceivers to effect a multipoint data-transfer configuration.

also receive interrupts, while other slaves go about their business uninterrupted by the data transfer. The address bytes thus control the data flow into a particular node. Indication of the end of a data block can come from either sending a data-length field at the beginning of the block or from the receipt of another slave or reserved address.

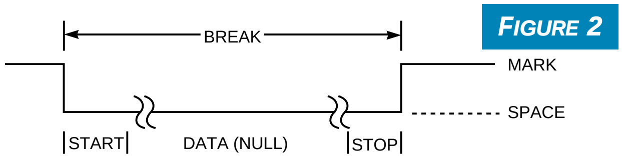
The crucial requirement for realizing the protocol is a means of distinguishing address from data bytes. You can effect this identification in many popular UARTs by using an obscure feature found in most UARTs: the capability to transmit and recognize (with an interrupt on) the break condition. This condition 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 stop bits (**Figure 2**). In this scheme, the whole data block (including address) from a master is sandwiched between break characters to form a data "frame" (**Figure 3**), and the address byte is recognizable as the one that immediately follows a break character.

The Turbo C program in **Listing 1** demonstrates the transfer of variable-size messages between two PCs (with 8250-compatible UARTs) using the method described here. **Figure 4** shows the 8250 register formats. The procedure works with most other UARTs. You can download the file from *EDN*'s Web site, www.ednmag.com. At the registered-user area, go into the "Software Center" to download the listing from DI-SIG #2193. A null-modem cable interconnects the PCs' COM ports. The destination PC accepts only the messages addressed to it. Note that, although the PCs here interconnect in a point-to-point manner, usually the stations interconnect using balanced RS-422 or tristate drivers in a multipoint configuration, as in **Figure 1**.

A global variable, Receive_Count, initialized to zero, handles frame reception. Initially, the protocol enables only receive-error interrupts. Each time the routine detects a break, the UART raises a receive-error interrupt, and the ISR (interrupt service routine) then enables the receive-data interrupts. On subsequent receive interrupts, if Receive_Count is zero, the ISR checks if the first address byte matches the station's address. If not, the receiver goes back to the initial waiting state, with the receive-error interrupts enabled and the receive-data interrupts disabled, such that the routine ignores the subsequent data bytes. If an address match occurs, the ISR stores the subsequent incoming data bytes in the receive buffer, with Receive_Count as index. If Receive_Count is nonzero when the break interrupt occurs, it is an end-of-frame break. Then the routine calls the frame-processing function, Receive_Count resets to zero, and the receiver again reverts to the initial waiting state.

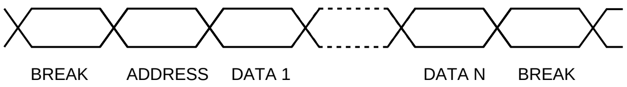
To transmit a break, the protocol sets bit 6 (set break) of the line-control register to one. The UART then takes its trans-

FIGURE 2



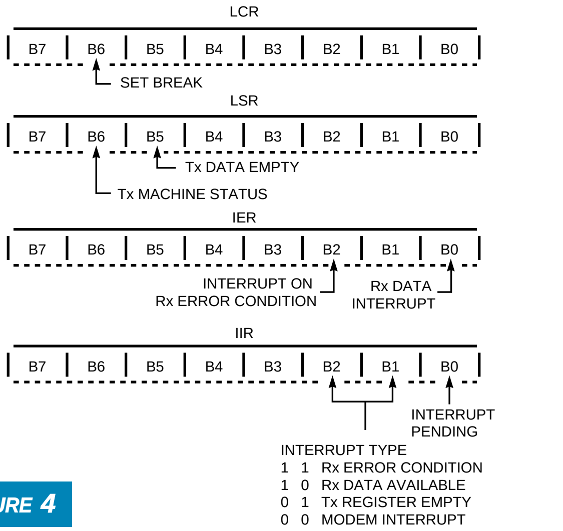
The ability to recognize the break condition is key to the master-slave transfer protocol.

FIGURE 3



This n-byte data frame shows the data block sandwiched between break characters.

FIGURE 4



These 8250 register formats demonstrate the multipoint-transfer protocol.

mission line low until bit 6 receives a zero. To make the duration of the break equal to one character-transmission delay, the routine transmits a null (00 hex) character. Bit 6 of the line-control register (transmit machine status) indicates when this delay is over; the break bit then resets. To enable detection of the break, bit 2 of the interrupt enable register (interrupt on receive error condition) sets during UART initialization. Bit 0, set to one, enables receive data interrupts. In the ISR, bits 1 and 2 of the interrupt-identification register indicate the interrupt type.

In this scheme, no CPU overhead is wasted examining each character to detect addresses/packet boundaries. Also, a slave must process only three interrupts per data packet transmitted on the bus, and blocks of data not addressed to the slave do not disturb it. Because the break is not a legitimate data character, it is data transparent; you can use it for binary-data exchange. The packet-boundary detection is immune to data errors. You can make it even more robust by including data-length and check-sum fields in the frame to enable error detection. You can also use parity error detection. Note that the method can support broadcast/multicast message transfer by designating some addresses for these purposes. You can also implement any-node-to-any-node communication by polling the master, as in the SDLC protocol. (DI #2193)

To Vote For This Design, Circle No. 413

LISTING 1–TRANSFER OF VARIABLE-SIZE MESSAGES BETWEEN TWO PCS

```
   #include <stdio.h>
   #include <stdlib.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);
   unsigned char Myaddr,Txaddr;



   void processdata(void) /* TO DISPLAY RECEIVED DATA PACKET */
   {
     int i;
     cprintf("\n\rRX Data > "); clreol(); /* Received data cursor */
     for (i = 1; i< Receive_Count; i++) /* Leaving out Addr byte */
   putch(rdatabuf[i]); /* Display received data */
     cprintf("\n\r"); /* New line */
     clreol(); /* Clear line */
   }

   void interrupt service_sio(void) /* ISR: TAKES CARE OF PACKET RECEPTION */
   {
     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 data */
    if((Receive_Count == 1) && (rdatabuf[0] != Myaddr))
    /* If First(Address) byte but no address match */
    {
outportb(INT_ENABL,0x4); /* IER; enable Only Rx Machine error int */
Receive_Count = 0;
    }
    break;
  case 3: /* Rx error (Break detect etc.) */
    inportb(DATA_PORT); /* Read Null char */
    if(((inportb(LINE_STS))&0x10) == 0x10)
    /* Break detected; Reading LSR Resets int */
    {
 if(Receive_Count) /* Complete Frame Over */
 {
    processdata(); /* Process the frame  */
    outportb(INT_ENABL,0x4); /* IER; enable only Rx Machine error int */
 }
 else outportb(INT_ENABL,0x5); /* IER; enable RX Data int also */
 Receive_Count = 0; /* Reinitialize for next frame */
    }
    }
    outportb(0x20,0x20); /* EOI to 8259 PIC */
    return;
  }

  void init_serial_io(void) /* TO INITIALISE SERIAL PORT */
  {
    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 com2 */
    outportb(INT_ENABL,0x4); /* IER; enable only Rx Machine error int */
    enable();
 }

 void SendBreak(void) /* TO TRANSMIT A BREAK OF ONE CHARACTER DURATION */
 {
    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 */
 }


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


      {
WAIT_TX_RDY(); /* Wait for Tx Ready */
outportb (DATA_PORT,packet[i]); /* Send next data char */
      }
      WAIT_TX_RDY(); /* Wait for Tx Ready */
      SendBreak(); /* Send END OF PACKET break */
      WAIT_TX_RDY(); /* Wait for Tx Ready */
   }



   unsigned char getaddr(char* mess) /* TO READ AN ADRRESS FROM THE CONSOLE */
   {
      unsigned char c, databuf[100];
      int addr,count = 0;

      cprintf("\n\r%s",mess); /* Prompt for input */
      clreol();
      while(1) /* Forever Loop */
      {
  if((c=getche()) == 27) exit(0); /* Exit if Escape key pressed */
  databuf[count++] = c; /* Get typed characters into the buf */
  if(c == '\r')  /* If Enter Key pressed */
  {
     if((sscanf(databuf,"%d",&addr) != 1)||((addr > 0xff)||(addr < 0)))
     { /* Read and check the string in the buffer for validity */
putch(7); /* Bell */
cprintf("\r\nError: Type in a number between 0 and 255");
cprintf("\n\r%s",mess); /* Transmit Prompt */
clreol(); /* Clear to end of line */
count = 0;
     }
     else break;
  }
      }
      return((unsigned char)addr);
   }


   void restoreint(void) /* FUNCTION WHICH DOES THE CLEAN-UP AT EXIT TIME */
   {
      setvect(0xc,(OldComHandler)); /* Restore int vector; 0xb for com2 */
      outportb(0x21,((inportb(0x21))|(0x10)));/* PIC mask word 0x8 for com2 */
   }
```