Implement a nine-data-bit UART on a PC

AUBREY KAGAN, WEIDMULLER LTD, MARKHAM, ON, CANADA

Many µCs, such as the 8051 and the 68HC11, can support a ninth data bit on the asynchronous serial port. This bit is useful in multidrop applications in which you can use it to denote an address on the serial bus, as opposed to data destined for a particular address. The UART used in IBM PCs (and clones) does not directly support this operating mode. However, through some software manipulation, you can add the PC to a serial bus and integrate it into a ninth-bit system, albeit with some limitations.

The method differs for data reception and transmission. As a result, the PC can work only in half-duplex mode. Because half-duplex communication is common practice on PC networks, this limitation is not a significant drawback. The technique also requires that the CPU check each incoming byte for the ninth bit. (You can usually configure a µC to generate an interrupt when the ninth bit is set.) For the PC to receive the nine bits, it is necessary to treat the ninth bit as a parity bit. Although it's impossible to read the parity bit in the PC's UART directly, it is possible to analyze the received data byte and determine what the parity should be.

If analysis reveals a parity error, then the value of the ninth bit is opposite to the calculated parity. If no error exists, then the value of the ninth bit is equal to the calculated parity. In the 16550 UART, the FIFO includes the three error bits with each data byte, so the parity error (or lack thereof) is always associated with the current data byte. It is possible, however, to disable the FIFO feature. The technique for transmission is slightly different. The 8250/16450/16550 UART has a forced-parity format (also known as a "stick" parity), in which you can set the parity to a one or to a zero. You do this by setting bit 5 (stick parity) and bit 3 (parity enable) in the UART's line-control register (LCR). The transmitted parity bit is then the logical inverse of bit 4 of the LCR.

In the sample code in **Listing 1**, address 0xff (with bit 9 set) is reserved and used to indicate the last byte of the transmission. The first byte of the transmission is an address, and it transmits with bit 9 set. The RS-232C port connects to an RS-232C/RS-485 converter, where the RTS line controls the direction. The code given here is not interrupt-driven, but you could implement it as an interrupt-driven routine. The code comprises three modules: background (back.cpp), serial procedures (serial.cpp), and memory declaration (mem.cpp). Note that mem.cpp declares one include file (mem.h) for the public memory. You can download the files from *EDN*'s Web site, www.ednmag.com. At the registered-user area, go to the Software Center to download the files from DI-SIG #2198. (DI #2198) e

To Vote For This Design, Circle No. 418

LISTING 1—BACKGROUND CODE FOR NINTH-BIT TRANSMISSION

```
//background program
//developed with Turbo C++
//operating under DOS

#include "mem.h"
#include <conio.h>

#define RTS 0x2

//prototypes
void setupUART (void);
void deAssert (int ControlPin);
void Assert (int ControlPin);
unsigned char SerialIn(void);
void UARTrx(void);
void UARTtx(void);
unsigned char UART_TX_clear( void);
unsigned char SerialOut (void);
unsigned int checksum (unsigned char NumberOfBytes);

void main (void)
{
unsigned int j;

     comport=1;
     //setting to COM1

     module_address=0xa;
     //PC address=10 decimal

     //other transmission constants
     setupUART();
     //initialise the UART

     capture_enabled=0;
     rx_pnt=0;
     //initialise variables

     Assert(RTS);
     //turn the RS485 buffer to receive

     while (1)
     {
          /*the actions are divided into several states as indicated by
the variable "phase".
          Phase=0- waiting for a complete serial message
          Phase=1- preparing a response
          Phase=2-wait for end of transmission
          Phase=3- wait for message to completely clear the UART (buffers
empty) & then
               re-enable reception (turn RS485 buffer around)*/

     switch (phase)
     {
          case 0:
               if (SerialIn())
               //checking for complete message received
               {
                    //now to process the input
                    phase++;
                    UARTtx();
                    //prepare UART to send
               }
               break;
          case 1:
               //prepare to transmit
               rx_buff[0]=0x0;     //destination address
               rx_buff[1]=0x13;    //response
               rx_buff[2]=0xff;
               //set last byte.

               number_of_characters=3;
               //variable for transmit routine
               rx_pnt=0;
               //intitialise the fetch pointer
               phase++;
               break;
          case 2:
               if (SerialOut())
               {//at the end of the message
                //bump on to wait for complete transmission
                    phase++;
               }
               break;
          case 3:
               //wait for message to clear
               if (UART_TX_clear())
               {
                    UARTrx();
                    phase=0;
               }
               break;
          default:
               break;
     }

     if (kbhit())
      {//terminate execution if any key pressed.
         break;
      }
}
```