DESIGN NOTE #042

AUTHOR: GLENN RICHMOGD

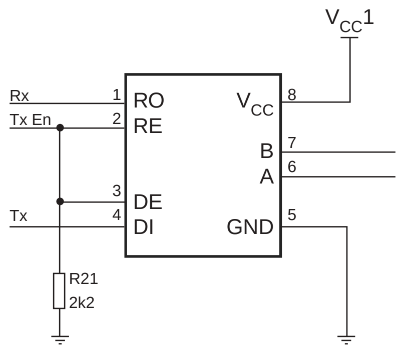
KEYWORDS: RS-485

Efficient Handling of RS-485 Timing Issues

Anyone that has had to deal with the timing issues involved in RS-485 communication knows that it can often be a time-consuming task. Although RS-485 communications have the advantage of being able to support multiple devices all running of the same line and require only two wires for a connection, the communications are only half-duplex. This means that there can be some interesting timing issues to overcome when it comes to switching between receive and transmit modes. If the processor switches from TX to RX mode too quickly, the entire packet will not be transmitted and will be invalid, and if it switches too slowly, the other unit may respond before it enters RX mode and the reply packet will be missed.

A common approach to overcoming this problem is to add significant delays in both the Master and the Slave units. This approach is shaky at best as it not only introduces unnecessary delays in the communications, but it can introduce uncertainty as to whether the packets will always make it through, especially when the length of the packets varies significantly. The way to get around this and have the fastest possible communications with the most reliability is to use the TX Complete interrupt in the AVR. To give an idea of the hardware being used, consider the circuit shown in Figure 1.

Figure 1. Cicuit Diagram of RS-485 Driver Configuration



The “Tx En” line switches the driver chip between transmit and receive mode (On = Transmit). The key is to switch this back to RX mode as soon as the final character is transmitted. To do this, we first populate a buffer with a packet, then begin the transmit sequence, and count the number of characters that have been completely transmitted. Since we know the size of the buffer to be transmitted, we can detect when the final character has been sent and therefore the earliest possible time that we can switch the line from transmit to receive mode. The source code for the putchar(), starttransmit(), initpacket() and uart_tx_isr() routines are shown below.

```
void initpacket()
{
  tx_wr_index = 0;
}

// UART Transmitter interrupt service routine
#pragma savereg-
interrupt [UART_TXC] void uart_tx_isr(void)
{
#asm
    push r26
    push r27
    push r30
    push r31
    in   r26,sreg
    push r26
#endasm
   if (tx_rd_index < tx_wr_index)
   {
   UDR=tx_buffer[tx_rd_index++];
   }else
   {
   switch_mode(RX_MODE);
   }
#asm
    pop  r26
    out  sreg,r26
    pop  r31
    pop  r30
    pop  r27
    pop  r26
#endasm
}
#pragma savereg+

// Write a character to the UART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
   tx_buffer[tx_wr_index++]=c;
}
#pragma used-

void starttransmit()
```

```
{
  tx_rd_index = 0;
  UDR = tx_buffer[tx_rd_index++];
}
```

The switch_mode() function seen in the code simply toggles the TX_EN line between on and off, depending on the required state. The function is shown below.

```
void switch_mode(char mode)
{
  if (mode == RX_MODE)
  {
    PORTD.2 = 0;
  }else
  {
    PORTD.2 = 1;
  }
}

Now, to send a packet, the sequence is as shown below:

void sendpacket()
{
// switch mode…
switch_mode(TX_MODE);
// initialise packet…
initpacket();
// populate packet…
putchar(0x01);
putchar(0x02);
putchar(0x03);
// send the packet
starttransmit();
}
```

The above function will now send the packet and switch back into receive mode and the earliest possible time. Note that the above code was written for the CodeVision compiler.