The Secrets of RS-485 Half-duplex Communication

Casper Yang, Senior Product Manager

support@moxa.com

RS-485 is a good choice for long distance serial communication since using differential transmission cancels out the vast majority of electromagnetic disturbances picked up by the signal. A simple RS-485 network consists of one master and up to 32 slave devices. Since RS-485 uses half-duplex communication—that is, the same two wires (D+ and D- shown below) are used for both transmission and reception—some means of controlling which side of the connection can transmit must be built into the system. In this article, we discuss the ADDC (Automatic Data Direction Control) concept and explain how ADDC works.

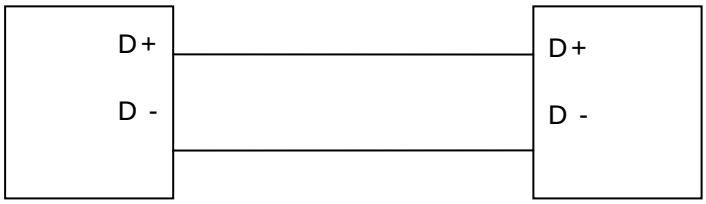

Fig. 1: RS-485 Half-duplex Communication

 Released on Sep 28, 2009

The most common way of controlling the transmit (Tx) and receive (Rx) direction is to use an RTS signal between the UART and the RS-485 half-duplex wiring. By adding a simple logic circuit (see Fig. 2), you can turn RTS on or off to switch the direction between Tx and Rx. That is, to transmit data you turn RTS on, and then you turn it off when the transmission is finished. Although the overall concept is easy to describe and understand, devising a precise enough timing mechanism can be quite a challenge.

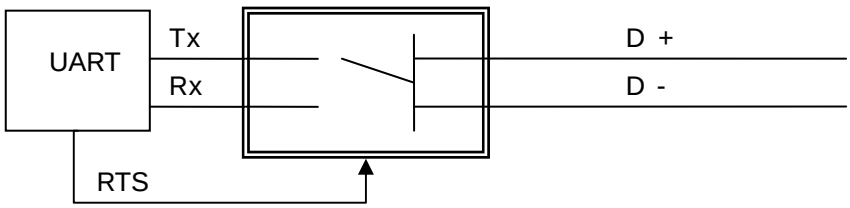

Fig. 2: Using RTS to Control Data Direction

In most cases, the RS-485 bus uses a master-slave architecture, which requires that each device on the RS-485 bus have a unique id. The master will send a command with an id and ask each slave to respond one by one. The default RTS state is off, which means that all devices are in the Rx state and are waiting to receive data (either a command or a response to a command) from one of the other devices. A typical scenario is as follows:

(1) The master switches to the Tx state, transmits a command to query a device, and then switches back to the Rx state and waits for a response.

(2) The slave whose id matches the id queried by the master switches to the Tx state, transmits its response, and then switches back to the Rx state.

If the master switches back to the Rx state too slowly, it will not receive the entire response. If the master switches back too quickly, the command will not be sent correctly. To control the timing properly, you need to know when the data was sent out.

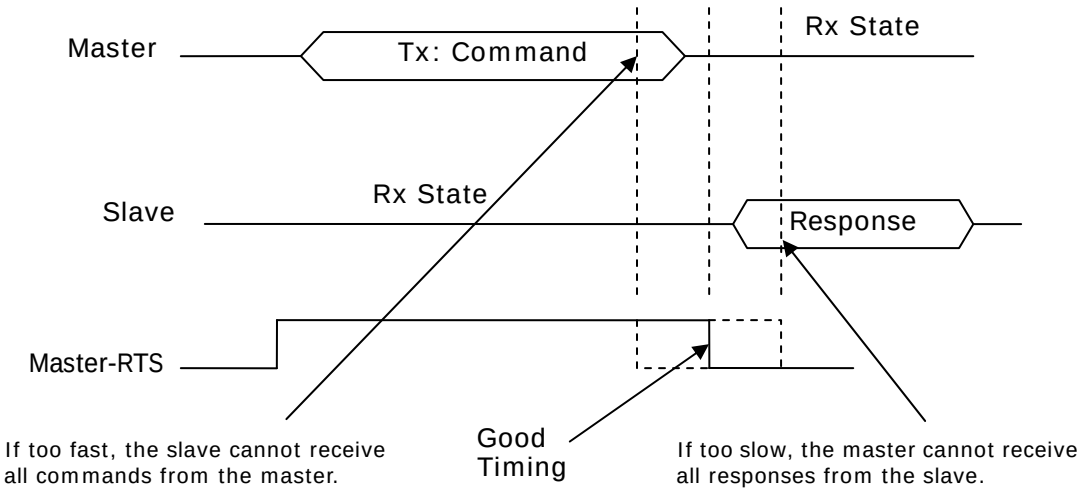

Fig. 3: Using RTS to Control Direction

UART Holding and Shift Registers—For Controlling the UART Directly

With the UART's Transmitter Empty Interrupt (TxINT) enabled, you may think it's okay to turn the RTS to off if no more data need to be sent. However, checking the TxINT value will only tell you that the holding register is empty, when in fact an additional byte could still be transmitting from the shift register. The UART's transmit shift register is used to transmit data bit by bit. Every time you put data in the FIFO, the UART will move data into the shift register automatically. To confirm that the shift register is empty, you need to read the Line Status Register (LSR) and check if both the Transmitter Holding Register Empty (THRE) bit and Transmitter Empty (TEMT) bit are set or not. If both are set, it is safe to turn the RTS to off.

UART Enhanced Mode with TTL Setting—For Controlling the UART Directly

To improve performance some advanced UARTs, such as the 16950 and Moxa's MU150/MU860, support "Transmit Interrupt Trigger Level" (TTL). This level defines when the UART needs to issue a Transmitter Empty Interrupt (TxINT) and cause the Interrupt Service Routine (ISR) to put more data into the UART. With this level set to a nonzero value, the UART can issue an interrupt even though some data is still queued in the Tx FIFO. To work properly in an RS-485 application, you need to set TTL to zero. This means that if you get a TxINT, the UART holding register and shift register are empty and you can turn RTS to off immediately if no more data in the buffer needs to be sent.

Applications in Win32 and UNIX/Linux—For Serial Application Programmers

In most cases, you do not need to control the UART manually, and instead just use the API provided by the operating system. You can use WriteFile() for Win32 systems, and write() for UNIX/Linux systems. Win32 provides the function RTS_CONTROL_TOGGLE to work with the RTS automatically. If fRtsControl is set to this value in the DCB structure by calling SetCommState(), the driver should turn RTS on automatically before sending the data and turn RTS off automatically when it finishes. Before using this approach, you should first make sure that it is supported by the vendor if you are using a serial expansion solution (which means that you are not using serial.sys, Windows' built-in serial port driver). If RTS_CONTROL_TOGGLE is not supported, controlling RTS manually will cause a lot of timing problems, and is not reliable.

For UNIX/Linux, the POSIX tty API does not support the RTS toggle function, and you need to control RTS manually. In this case, you can call tcdrain() to wait until data is sent and turn RTS off. The problem is that tty drivers only check the driver buffer and UART holding register. Unfortunately, there is no way to confirm that the shift register is empty. If you really want to use RS-485 in UNIX/Linux with RTS control, you need to modify the driver or call your vendor for support.

Working with Virtual Serial Ports—For Serial Application Programmers

If the serial port is a "virtual" port, such as is created by a USB-to-serial or Ethernet-to-serial product, it's not easy to know when all the data has been sent out because the driver needs to work with the firmware in the product and not a real UART. For virtual serial drivers, the write operation will be finished immediately when the data is moved to the local driver buffer. This is good for performance but is not good for RS-485 RTS control. If you turn RTS off immediately when the write operation is finished, the data will still be transmitting, which will cause problems. In this case, in Win32 you can use RTS_CONTROL_TOGGLE if it is supported by the driver. If not, you need to see if the driver supports an advanced option (such as Moxa's "classical mode" in UPort and NPort) for confirming that all data will be sent out before the WriteFile() is returned. For virtual serial ports it is impossible to control RTS manually without RTS_CONTROL_TOGGLE or classical mode supported in Windows/UNIX/Linux.

Controlling RTS Automatically—For Everyone

Controlling RTS by software (driver or application) can give rise to a number of problems, and consequently many hardware vendors solve this problem by controlling the direction automatically. This means that the RS-485 hardware can switch the Tx/Rx direction automatically, such as is done with Moxa's ADDC (Automatic Data Direction Control) function. It is compatible with existing software and if you want to transmit commands over the RS-485 bus, you can just send, instead of worrying about controlling the RTS signal. The hardware will detect the action and switch to the Tx state automatically. Note that some converter products require users to configure the baudrate first. Please read the manual of your product for detailed information.

Conclusion

A good way to simplify RS-485 implementation is to choose a product that supports ADDC. With ADDC you do not need to waste time modifying your programming to match the timing. You can keep your existing programming as is, and rest assured that it works with ADDC. Controlling the RTS on/off process will be ignored by ADDC, and all you need to do is configure the hardware properly.