Print Date: 25.06.2003

White Paper on Buffer Overrun

Version: 0.92

Brainboxes Limited, Unit 3C, Wavertree Boulevard South, Wavertree Technology Park, Liverpool, L7 9PF, UK
Tel: +44 (0)151 220 2500 Fax: +44 (0)151 252 0446 Web: www.brainboxes.com Email: sales@brainboxes.com

Table of Contents

1. Introduction

This white paper presents the issues relating to the buffer overrun during serial communication using Brainboxes Serial Solutions cards. In the following sections, we discuss the background on components of serial communication, about buffer overrun and system limitation. We also discuss how adjustments can be made to improve performance on serial communication throughput.

2. Background

2.1. Hardware UART

Asynchronous RS232, RS422 and RS485 communication most often uses a UART (Universal Asynchronous Receiver Transmitter) to transmit and receive the data across serial cables. All Brainboxes Serial Solutions cards have UART chips, which control the data communication at the hardware level. Inside the UART chip, there is a FIFO buffer where the incoming data for the software driver is stored. FIFO sizes are 1, 16, 64 or 128 bytes depending on the UART type. The FIFO is used to improve the throughput of communication between two serial ports by buffering data. There are separate receive and transmit FIFOs.

2.2. Software Driver

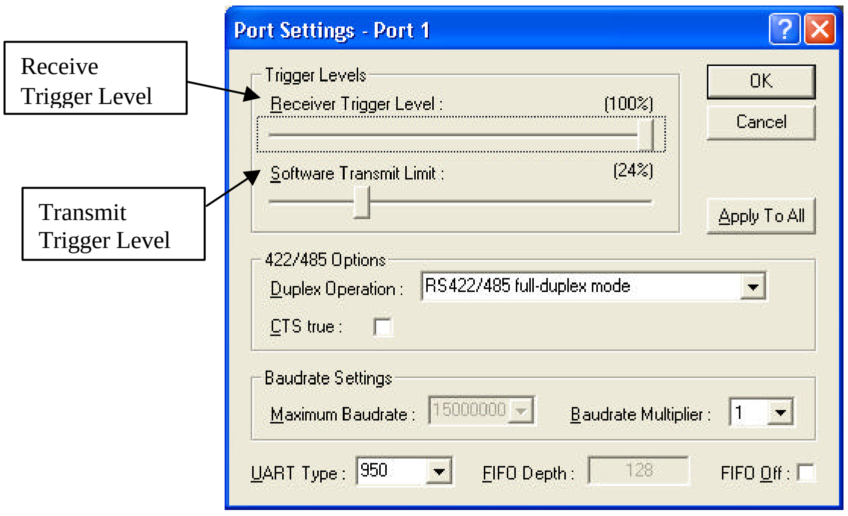
Brainboxes' Serial Solutions driver uses interrupt driven I/O (input/output). Among the events, the UART will issue an interrupt whenever data is available to be received from it. This then provokes the OS to invoke the appropriate driver interrupt service routine (ISR). It can also notify the serial driver by interrupt when the UART is ready to transmit or the line status of the serial communication has been changed. Due to the system design issues, Brainboxes PCI serial cards use the same interrupt for different ports on the same serial card. That is, every serial port on the same card shares the particular interrupt assigned by the system.

The received and transmit trigger level value controls how much data should be queued before an interrupt should be issued. When the FIFO is enabled, trigger levels are applied. This tells the UART how much data to queue before sending an interrupt to the OS.

The received trigger level value sets the received threshold level for the UART. The RDA (Received Data Available) interrupt is issued when the received FIFO has reached the threshold level of incoming data. If the total amount of data received is less than the trigger level, then, after 4 character times a CTO (Character Timeout) interrupt is generated. Setting this received trigger level value to higher causes less interrupts to be generated per data bit. However, care should be taken when setting this threshold. If 100% is selected, the FIFO will fill before an interrupt is generated, it will be sometime (interrupt latency) before the driver's ISR will be able to read the data. If, this interrupt latency is larger than 1 character time then the FIFO will overflow. At higher data rates, this interrupt latency becomes critical as it may extend to multiples of character times and thus the receive threshold must be set significantly lower than 100%. This may not actually increase the ratio of interrupt per data as there is likely to be more than 'threshold' bytes of data in the FIFO by the time the interrupt latency has expired.

Like the received trigger level, the transmit trigger level value sets the transmit threshold level for the UART. The THRE (Transmit Holding Register Empty) interrupt is issued when the transmit FIFO level falls below the threshold level during transmission.

The following diagram shows the dialog for receive and transmit trigger level in the Serial Solutions driver.



Trigger Levels
Receiver Trigger Level : (100%)
Software Transmit Limit : (24%)

422/485 Options
Duplex Operation : RS422/485 full-duplex mode
CTS true :

Baudrate Settings
Maximum Baudrate : 15000000
Baudrate Multiplier : 1

UART Type : 950
FIFO Depth : 128
FIFO Off :

3. What is Buffer Overrun and Why?

There are two different types of buffer overrun: hardware and software buffer overrun. Both types of buffer overrun are more likely to happen when transferring continuous streams of large amounts of data across the serial port using high baud rates e.g. >1M baud.

A hardware buffer overrun occurs when the UART is unable to manage putting/placing incoming data into received FIFO because there is no space left in the FIFO. This means the incoming data is arriving faster than the serial driver can read characters out from the received FIFO. Or, the interrupt service routine is not handling the data efficiently. This type of overrun may occur when using high baud rates because of the bus or system limitations (discussed in the following section). The UART indicates hardware buffer overrun by issuing a RLS (receiver line status) interrupt when the line status register changed. In this case, the user application is notified of communication error by a CE_OVERRUN error code but not all applications will react to this error.

A software buffer overrun occurs when the software buffer in serial driver is not big enough to handle buffering of incoming data from the UART. The serial communication applications set the receiving buffer size for user application. The serial software driver transfers incoming data, which is saved in the application's internal system buffer. Software buffer overrun occurs when the driver cannot transfer data to the user's receiving buffer because it is full. This type of error is issued from the serial software driver not from the UART. Inefficient handling of the user application may cause this type of overrun. In this case, the user application is notified of communication error by a CE_RXOVER error code but not all applications will react to this error. This kind of communication error may be prevented by increasing the application's receive buffer size.

4. System (PC) Limitations

Brainboxes PCI serial cards use a PCI target (slave) chip capable of operating on the 33MHz 32-bit PCI Bus only. The 33MHz PCI Bus bandwidth must be shared across all devices on the bus. The extent of this sharing depends mainly on the motherboard/controller chip architecture.

The implementation of our PCI serial cards means that it is not uncommon for each register read from any given UART on that card, to take up to 30 PCI clock cycles to complete. All standard UARTs have an 8-bit data bus so each potential 32-bit read only delivers 8 bits of real data.

Due to bandwidth sharing, it is not uncommon to see less than 30% of the PCI bus available bandwidth on average to be available to PCI target devices, even in a 'lightly loaded' system. This may be worse when, say, a network card has taken control over the bus to perform an extended transfer of some nature.

So …

33 MHz x 30% ˜ 10 MHz (effective bandwidth)

10 MHz / 30 clock cycle ˜ 0.3 MHz (In principle, one data byte can be transferred per clock cycle.)

i.e. effective max. data rate = 0.3 M bytes / sec

At 8 data bit, no parity and 1 stop bit each byte actually has 10 bits of information sent across the wire.

Sending 1 byte per second = 10 bits/ sec

So, this 0.3 Mbytes /Sec with 10 bits per byte ✍ 0.3 x 10^6 x 10 = 3 Mega baud.

5. Conclusion

Under highly loaded conditions the highest data rate one might sustain by using our current product is 3-Mega baud. For example, our 3 Velocity RS-232 ports are operating at maximum 921-K baud. By transferring data stream over all ports concurrently, it will be resulting in the total 3Mega baud (approx. 1Mega baud on each port). This fits well with our own experimental data.

Other recommended adjustments to prevent overruns are:

?? Reduce the baud rate: slower per data bit, but lack of overrun errors sustained throughput.

?? Use a faster processor system: reduced interrupt latency, but not guaranteed because it may still have bus limitation.

?? Handshaking: it needs implementation at each end of the link. But enables some degree of 'protocol' to the sent flow of data. Many of our UARTs have hardware auto flow control so that the latency of this operation is minimised. We do not recommend use of software flow control as this increases latency times.

6. Performance Adjustment

In serial communication, the communication speed is measure by bits per second (bps) or baud rate. It means the actual physical medium transmission rate across serial cable. The data throughput is the sustained rate of data that can be achieved during extended data transfer.

By adjusting the receiver buffer trigger level it may be possible to reduce the frequency of overruns in the system. For example, in Pentium II 350 MHz with 160 MB RAM machine, by adjusting receiver trigger level to 50-60% may show improvement for transferring data streams over 2 serial ports using 921600-baud rate concurrently.

7. Acronyms

CTO	:	(Character Timeout) An interrupt generated by the UART when the total amount of data received is less than the trigger level and no more character is received after 4 character times.
FIFO	:	(First In First Out) A buffer in UART where incoming and outgoing data for the communication software driver are stored.
Hardware buffer overrun	:	Hardware buffer overrun occurs when the UART is unable to manage putting/placing incoming data into the received FIFO.
Interrupt driven I/O	:	An interrupt will be issued by the UART whenever data is available to be received or the UART status is changed.
RDA	:	(Received Data Available) An interrupt generated by the UART when the received FIFO has reached the threshold of incoming data.
Received trigger level	:	A value set for the received threshold level for the UART.
RLS	:	(Receiver Line Status) An interrupt generated by the UART when the line status register is changed.
Software buffer overrun	:	Software buffer overrun occurs when the driver cannot transfer data to the user's receiving buffer.
THRE	:	(Transmit Holding Register Empty) An interrupt generated by the UART when transmit FIFO is empty.
Transmit trigger level	:	A value set for the transmit threshold level for the UART.
UART	:	Universal Asynchronous Receiver Transmitter.

Version History

Version	Date	Author	Checked By	Comments
0.90	6/6/2003	Michael Wilcox, Maung Bandu	Michael Wilcox, Gavin Jewell	
0.91	12/6/2003	Michael Wilcox, Maung Bandu	Gavin Jewell	Addition of introduction section. Rewrite background software driver section. Add conclusion section and set recommendation. Made other modifications.
0.92	12/6/2003	Michael Wilcox, Maung Bandu	Michelle Robert	Made some modifications on writing.