

## Re: Serial Port CE\_OVERRUN errors

---

*Source:* <http://coding.derkeiler.com/Archive/Delphi/comp.lang.pascal.delphi.misc/2006-02/msg00110.html>

---

- *From:* "Paul E. Schoen" <pstech@xxxxxxxx>
  - *Date:* Tue, 7 Feb 2006 04:11:31 -0500
- 

"Hans-Peter Diettrich" <DrDiettrich@xxxxxxxx> wrote in message  
<news:44qd2qF3flrsU2@xxxxxxxxxxxxxxxxxxxx>

Paul E. Schoen schrieb:

Since I am sending the data at exactly 4800 characters per second (in  
pairs), speeding up the baud rate just puts more space between the

pairs,

hopefully to allow the USART to do its buffering before the next start

bit

occurs.

The UART has enough time to do the internal buffering, for any supported  
baudrate. Problems can arise only from a slow response of the next  
stage, the system that retrieves the data from the receiver.

I plan to recommend that my customers use only WinXP and a 1 GHz or

better

processor, which will not impose any hardship.

I wonder how we could perform data acquisition in 1988, with 9600 Baud,  
with the first portable 8088 PC, running at 0.004 GHz, with online  
graphics of the received data, and writing to diskettes. We only had to  
use a RAM disk later, because diskette operation was unreliable in a  
moving car (Porsche ;-).

IMO there's something wrong with your data acquisition machinery or

## Re: Serial Port CE\_OVERRUN errors

software. Imagine what amount of information comes in from an Internet connection, and can be handled even by older systems.

DoDi

The problem is that I must read and process the incoming data essentially in real time. My previous system, running on MSDOS and using the parallel port, will run on an old IBM laptop which probably has a 12 MHz 80286. It would even work without a math coprocessor. The data acquisition rate was 2000/second. I speeded up the usual 18.2/sec system clock to 2000/sec, and used an ISR to perform the read from the previous conversion, and then fired the next conversion. I had an ISR overrun detector which sometimes indicated a new interrupt before the processing had completed, so I just allowed that conversion to go unread. This problem actually got worse with faster machines, especially when EMM386 or HIMEM.SYS were loaded. The memory managers apparently could override my Disable Interrupt commands, or at least had a higher priority interrupt (maybe NMI), that sometimes would cause several samples to be missed. Then, even faster machines started working better, but still showed some ISR errors unless the memory managers were disabled. Most customers just booted their computers with plain vanilla MSDOS and ran our software. However, many computers, especially portables, still had problems, probably because of BIOS level interrupts for power management, display, etc. Eventually, as of a few years ago, the need for a Windows based system became apparent.

The old system got around slow processing by simply dumping the raw data into buffers, to be fully analyzed later. However, the ISR did have to determine when each of four AC current pulses started and stopped, in real time, which is not as simple as just setting a threshold, since it is AC. I also had to implement a pretrigger function, so I could go back and read and store data before the actual detection of current. The display was updated about 5 times a second, and it processed the stored readings as far as it could. On slow machines, the test would be over (a complete test takes about 5–10 seconds), and then you could watch the readings catch up. A 40 MHz 386 machine kept up with the readings in real time.

For the new design, my partner in this endeavor was worried about losing data when the Windows operating system did its multitasking. I was prepared to write the code for the hardware system so that it would process all of the readings and just send data to the application for continuous current value every 200 mSec, and calculated current values and times for each operation as it occurred. This would have greatly simplified the communications part of this project, but would have greatly increased the complexity of the hardware and PIC code. It is also highly desired to store the waveform for storage and playback, so this would have required additional memory beyond the 4k Bytes or so on most PICs, and even taxing the resources of microcontroller products like the Z180 and Rabbit products. My early experiments proved that the serial port of the PC should be able to handle the continuous stream of data, which would also provide a highly accurate timebase which could be calibrated with the clock in the external

## Re: Serial Port CE\_OVERRUN errors

hardware, rather than relying on the PC system clock as before.

The Ortmaster software went through many stages of improvement, and I learned more about serial port components and their usage. The SerialNG component seemed to give the least trouble, but errors would always seem to occur. We had some major problems with commercial USB to Serial converters, which sometimes showed framing errors or others, yet the data seemed OK. I got a SiLabs CP2103 USB to USART bridge kit, and it did not show those errors, but it would cause the application to appear to hang up for about 1 second intervals. I found this to be caused by timeout settings in the device which waited until most of its 4096 byte buffer was filled before spitting it out the USB port. I made the timeout shorter, and now it sends the data just about as soon as it is received. This probably contributes to the high overhead of the USB implementation, as possibly each byte might be sent in a USB packet consisting of maybe a dozen bytes, so that would soon cause a bottleneck even at USB baud rates, which I think are only 100k to 500k for full speed. What probably happens is that the data packets grow in size until the USB port catches up, and then short packets are sent again. It would be ideal to set the packet size to something like 480 bytes, or 10 bursts per second. Maybe if I go to a direct USB implementation it will be more efficient, but the major changes in hardware, firmware, and software design are daunting, and I would lose the versatility of being able to use a native serial port.

Anyway, I am satisfied with the present level of performance on my XP machine, using this CP2103 device, and I am becoming more confident in the basic stability of my Delphi software. It's not good OOP, it ain't pretty, but all functions seem to work. I plan to look at the overhead in my 200 mSec update procedure, which I should be able to do by looking at the number of characters waiting in the CommBuffer when it has finished processing all the characters that were there at the beginning. I could even display the ratio of the amount of time spent in the loop to the 200 mSec interval as a measure of overhead. This would, of course, also include other processes that are occurring with task switching. I may be able to alert the user that he or she needs to shut down other applications or get a faster computer, or run the risk of getting errors and spoiling the test. Since this testing involves sending up to 600 VAC at possibly several hundred amperes through a utility line protection device, the operator should not be fooling around with multimedia movies or games!

Now it's time to move on to make the next version of the PC board, with a built-in CP2103 chip and possibly a multi-port hub. I have not heard anything in over a month from my partner and his programmer on the progress of the TCC program. It's mostly a simple database application that checks my test results against manufacturer's specs, so I would think it would be maybe a two week project in Delphi. They have decided to use VB.NET, and it's been in development for almost two years now with little progress I can see, and still lots of unhandled exceptions.

If you are still reading, I thank you for your interest.

Re: Serial Port CE\_OVERRUN errors

Paul E. Schoen  
www.pstech-inc.com