Fast RS-232

by AgentFriday

Intro

One of the nice thing about the older processors is that they pretty much all operated deterministically in . This just means that if you start from a known state, and supply exactly the same sequence of inputs, the system will arrive at the same end state, every time.

Although I have seen claims that the built-in RS-232 routines can communicate reliably at 300 baud, even as high as 1200, those making the claims must not have used it for anything more that slow typing. Heck, I couldn't even do that at 75 baud without seeing the occasional error!

→ the builtin (kernal) routines can do 1200 baud reliably. 2400 baud can be done reliably with custom routines. anything more needs additional hardware, either the hack published by daniel dallmann (can do 9600baud, needs special drivers) or an uart cartridge such as swithlink (which effectivly does about 14.4k baud on a stock machine). 38.4k baud i dont see working on a stock machine anytime soon, even with swithlink. /gpz ←

> Groepaz– I am quite willing to be proven wrong. Up for the challenge? I can share my tests with you if you're interested.

.

OK, here's what I know… A friend and I both build our own RS-232 driver circuits and wrote our own programs to test communication (using whatever info we could find on the topic). No matter what we connected to (internet, terminal on PC, another C64) or how we tweaked the code, we both saw the same sorts of errors–corrupted data (single/multiple bit errors) and missing characters (framing errors). After an extensive search we found an article documenting the various errors & vulnerabilities in the Kernal routines, and it also provided replacement routines to fix the problems. Plugged the new code in and it all worked like a charm, at 2400 baud.

I would really like to see a working demo of the built-in RS-232 functioning properly at 1200. Or even 300.

STATUS UPDATE

My first test of the code failed because the assembler I was using (C64ASM) generated absolute mode opcodes for all of my zero page operations. This stretched out the timing so much that–best case–bit 7 was sampled within 2 or 3 cycles of the stop bit transition, and worst case it was a couple usecs outside the bit frame. Amazingly, about 1/4 of the characters came through clean, and there were no dropped characters or framing errors as far as I could tell. I interpret this to mean that at least 18-19 of the 26 cycles in each bit are clean and stable. So I don't the circuit I'm using gives any reason for concern.

I'm currently making transition to a new assembler. Since I've adopted cc65 as my primary development tool, I will be giving ca65 a short. Hopefully I'll have the kinks worked out soon. I'll post my code when it works.

– AgentFriday

You'll want to check out over5 by Daniel Kahlin, which does 38.4kbps through the raw user port. I used the same concepts and got it working at 56kbps as well. - White Flame
Thanks White Flame. I've heard unsubstantiated rumors of a 56k implementation–so we meet at last. :) My 38.4 code is nearly functional.