Help understanding odd USB-Serial test results (Teensy 4.0)

Thread: Help understanding odd USB-Serial test results (Teensy 4.0)

Tags: *None*

sleeper said:
02-15-2020 09:59 PM

Help understanding odd USB-Serial test results (Teensy 4.0)

I'm working on a project which involves a signal chain through multiple Teensy4's on a usb hub, so I started by testing to see if I could meet my bandwidth and latency requirements.

I wrote a usb-serial echo sketch for the teensy, deployed it to 4 teensies on a usb hub, and then wrote a low-level mac program to send N bytes of data to each serial port and also receive N bytes of last-cycle's data in a round-robin fashion on a precise 1ms interval.

I then went to tune N-bytes per cycle to find the limits. I was finding that the buffer was actually being corrupted in transit before any byte shortages/underflows. It was also a very intermittent corruption, sometimes not showing up until the 900th cycle, often much sooner however. The corruption was of the form of 64-byte chunks being rearranged or copied. I sent a buffer of 32 1's followed by 32 2's etc and what I'd get back would be made of the same values but in the wrong order, or with duplicated chunks.

So with buffer "N" up to 384 there we no such complications, it just worked. I tried increasing NUM_USB_BUFFERS to 24 (24*64 = 1536 bytes) but no improvement. I also switched form Serial.read to usb_serial_read as it looks like a more efficient call.

I went through many iterations of the serial-echo sketch before getting to this simple change which makes the problem go away:
To get stability with sizes up to 960, I have to insert this "if bread < N" into the code.
I was testing multiples of 192 so 960 was my upper limit before seeing underflows at the mac serial read.
That bandwidth is actually adequate for my needs, but it bugs me not understanding what's going on.

Code: [View]

```
    if(bread < 960) { return; }

    while(bwrite < bread)
```

defragster said:
02-15-2020 10:25 PM

On Teensy 4.0 the 480 Mbps USB-Serial will be using 512B transfers AFAIK.

Not sure how that relates except the edit above for 24*64 wouldn't be right?

The T_4 USB stack has been updated and improved - but perhaps not yet perfected. It has been some weeks and a couple of TDuino releases since changes - before that I recall something about large transfers.

Not sure if those thread notes might be found for reference or if they still apply - it is also possible the high throughput is messing up on the MAC end?

If a sample can be made that exhibits the problem - especially on MAC I'm sure Paul would happily put it on his list when he can get back to that.

Is this done with TeensyDuino 1.51b1? No recent USB changes as noted - but as of now that is what the test and change would be against for a fix.

sleeper said:
02-15-2020 11:01 PM

This is teensyduino 1.50, downloaded fresh 2 days ago.

Seems very unlikely (to me) to be an issue on the mac end as I'm using posix read() and write() in a very trivial manner and it's reporting sending and receiving the correct number of bytes.

The chunk-wise re-arranging/copying looks like some form of ring-buffer corruption with overflow or underflow.

One speculation I had was the while loop to synchronously write back was taking so long the internal input fifo was overflowing, and my if condition causes it to wait for a lull in input (in between full blocks sent by mac) to then send back.

I was looking at the 480 high speed mode, which seems to automatically enable in usb.c based on some runtime information:

Code: [View]

```
if (status & USB_USBSTS_PCI) {
        if (USB1_PORTSC1 & USB_PORTSC1_HSP) {
                //printf("port at 480 Mbit\n");
```

Based on my data starting to underflow at 1100 bytes / ms, that feels a lot more like 12Mbit serial mode was active.

defragster said:
02-15-2020 11:47 PM

MAC should be fine if net throughput is under 15 MB/sec and running high speed.

Surprising if not running high_speed you can check that value in sketch it seems with :: extern "C" volatile uint8_t usb_high_speed; // non-zero if running at 480 Mbit/sec speed
Code: [View]

```
        }
// …
}
```

Paul works on MAC so your source there should work for him.

TD_1.50 is not old - just that 1.51 has to come out for updated IDE 1.8.12.

PaulStoffregen said:
02-16-2020 12:01 AM

> Originally Posted by **sleeper**
> but it bugs me not understanding what's going on.

Me too.

Any chance you can give me enough info & code to reproduce this here. I have 5 macs here that I use for testing and developing Teensy's support for MacOS. They run MacOS versions 10.15, 10.14, 10.13, 10.12 and 10.7. I do not have any way to test on old MacOS 10.8 to 10.11.

There's probably no point using 1.51-beta1 if you're already on 1.50. The only significant change is support for Arduino 1.8.12.

Nominal Animal said:
02-16-2020 01:40 AM

But.. are you sure you are using POSIX read() and write() correctly?

A very common error I see is using `if (read(ttyfd, buffer, bytes) < 0) error();`, assuming that `read()` will always read the specified amount of bytes or fail; it does not. Character devices and sockets can return a short count, *and often do*. If you too made this error, you'd see stale content at the end of your buffer from time to time.

Also, I'd put the ttys into nonblocking mode (via `fcntl()`) and use `select()` to see which ones are readable and/or writable; and noting that an attempt that fails with -1 with `errno` matching `EAGAIN`, `EWOULDBLOCK`, or possibly `EINTR` (if an interrupt was delivered to a handler using this thread that was not installed with the `SA_RESTART` flag) is not an error; it just meant that the kernel *thought* a read/write should succeed, but was (temporarily!) incorrect; just ignore these as if `select()` had not marked them readable/writable. I'd then attempt a read/write to each descriptors that were returned as readable/writable (separately, i.e. allowing a read and a write to the same descriptor in the same pass).

This said, while I do have two Teensy 4.0's, I haven't tested them yet in this manner. So, I am **not** claiming your POSIX C is at fault. I am only saying that I have seen way too many programmers make the above mistake about POSIX low-level I/O, and only want to make sure this is not the case here, as it *could* explain the behaviour you are seeing.

sleeper said:
02-16-2020 01:56 AM

Modified defrag's sketch a bit, and it seems it is in fact going into 480 mode, so thats good I guess.

A little more about my setup: MBP 16 OSX 10.15
Tried with a usb-c dock-style usb adapter to just one T4.0, tried with a mini usb adapter, and tried with the mini adapter to a 4-port usb3 hub.
Don't have adapter-less option, but all my adapter behave the same.

The code compiles to a binary you call like:

Code: [View]

```
ChainTest 6 /dev/cu.usbmodemWHATEVER /dev/cu.usbmodemWHATEVER2
```

The buffer size is 6 * 192 = 1152 in that example, which fails the test cases in the mac code.
If you pass it a single serial port, it runs the test with just one teensy, which I'm still seeing the same bandwidth cap somehow.

Read/Write drop outs mean the posix IO did not copy all bytes, so when that is non-0 the buffer error is expected sympathetic error.
The weird cases are when there are no read/write drops logged but the buffer is still corrupted.

So I'll just dump all the code on ya, it's fairly readable.

Here's a few hundred lines of code...

Code: [View]

```
#pragma once

#include <stddef.h>
```

Code: [View]

```
#include "Serial.h"

#include <string>
```

Code: [View]

```
    auto target = std::chrono::high_resolution_clock::now();
```

Last edited by sleeper; 02-16-2020 at 02:09 AM.

sleeper said:
02-16-2020 02:26 AM

Update:

Switched the mac code to use blocking IO and loop over each read/write until all bytes copy, and it seems I'm getting substantially better bandwidth, up to 13*192=2496 bytes in and out per cycle over 4 T4's

Previously was just logging failure if non-blocking IO did not copy all bytes, and there's probably a mac buffer size like 1024 that was capping me.

But I'm still running into the mysterious case where the fd IO is reporting everything good but my buffer is still getting corrupted in transit.

Nominal Animal said:
02-16-2020 07:18 AM

Using
Code: [View]

```
#include "usb_serial.h"
#define  MSGLEN  512
```

on two Teensy 4.0's (using just-installed Teensyduino 1.50 on Arduino 1.8.10), in parallel on the same USB host port (hanging off a cheap USB 2.0 hub), using 512-byte messages and my own async benchmarker, I get 2.5 - 2.8 Mbytes/sec bandwidth on each, with a quarter-millisecond median round-trip latency on Linux (HP EliteBook 840 G4, running Linux 5.3.0-28-generic #30~18.04.1-Ubuntu SMP on x86_64), and no errors in packet contents. The host side generates a new pattern for each message using Xorshift64*.

There is obviously host overhead involved, since using 1024-byte messages on the host side (but the above sketch using 512), I get 3.6 - 4.0 Mbytes/sec bandwidth, still with quarter-millisecond median round-trip latency, but occasionally one of the Teensy 4.0's will glitch, returning incorrect content for some of the messages, and faster than normal.

Using even longer messages causes one or both Teensies to miss some incoming data, so the benchmark fails. This is almost consistent at 2048-byte messages, and quite consistent at 4096-byte messages on the host side (but the above sketch still using 512).

Note that "message" is kind of a misnomer here, because both the sketch responds immediately with any data it receives, and the host program does nonblocking I/O, and writes and reads simultaneously; the host program does not send the full message and then read, but reads during writes also:
Code: [View]

```
#include <unistd.h>
#include <fcntl.h>
```

I used gcc-7.4.0 (`gcc -Wall -O2 benchmark.c -o bench`) to compile the host-side program. Run without arguments to see usage; the test cases I mentioned above are via `./bench 512 10000 /dev/ttyACM*` et cetera.

My opinion is that the USB serial stack on Teensy 4.0 misses some incoming packets (or parts of the

incoming stream), and/or fails to send some outgoing packets (or parts of the outgoing stream). I'd need to wire a small display (OLED, maybe) to one or both to see the received/sent byte counts on the Teensy 4.0 end to verify, but I'm lazy, and because I don't actually need this to work right now myself, I'm hoping @PaulStoffregen will have time to look through/debug the USB serial stack on the Teensy 4.0 instead.
