

Re: In-tree version of new FireWire drivers available

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-01/msg06436.html>

- *From:* Pieter Palmers <pieterp@xxxxxxxx>
 - *Date:* Thu, 25 Jan 2007 07:03:52 -0300
-

Kristian Høgsberg wrote:

On 1/24/07, Pieter Palmers <pieterp@xxxxxxxx> wrote:

Kristian Høgsberg wrote:

Changes since the merge into the linux1394 tree include:

- gap count optimization
- full bus management
- loopback for async requests to the local node
- a bug fix for a problem exposed by VIA 6306 controllers
- a typo fix from the bitfield -> mask+shift conversion.

Kristian,

What is your ETA on a the completion of the isochronous interface?

I'm hoping I can wrap this up within the next 1-2 weeks. So far I've been thinking about how to use the dualbuffer receive mode, and it turns out that it's a little tricky. It's nothing that can't be worked around, but I haven't yet made up my mind on the design.

Just to recap, the dual buffer receive mode, as described in section 10.2.3 of the OHCI spec allows us to set up DMA so that a fixed, quadlet aligned amount of header data can be appended into one buffer and the rest is appended into another buffer. This allows us to strip out the ieee1394 iso header as well as the iec61883 header for those protocols. That way DMA can assemble a complete DV frame without CPU intervention, strip off audio headers or just strip the iso header like video1394 does, which is sufficient for IIDC cameras. So this has the potential of actually replacing video1394 while at the same time generalizing the iso header stripping feature to be useful for iec61883 based protocols.

The problem is that the dual buffer descriptor stops appending when *either* the header buffer or the payload buffer fills up. When the

Re: In-tree version of new FireWire drivers available

payload buffer fills up, and this is what we'll typically hit, the last packet will continue into the buffer setup in the next descriptor, and the contents will probably straddle the two buffers. Each buffer will be a page in memory and since we map those into user space linear memory, that's not a problem.

I'd like to make one note here:

We should have a way to use smaller DMA buffers than one page size. If I remember correctly, the page size on my system is 4096 bytes, being 1024 quadlets. If we assume a 4 channel audio stream, this corresponds to 256 audio samples. This means that the controller generates an interrupt every 256 samples, making that we can achieve a latency of 512 samples at best. This is unacceptable in a pro-audio environment.

The current stack exhibits this problem, and I solve it by recalculating the max packet size, based upon the stream composition (i.e. expected packet size) and the requested audio buffer size, such that the interrupts are generated at a high enough frequency.

I'm not a kernel hacker, but when looking through the code I had the impression that smaller DMA buffers were possible (aren't smaller buffers used in packet-per-buffer mode?).

However, the other case is when the header buffer fills up. In this case, the DMA engine moves on to the next descriptor in the list and starts from new in the payload buffer from that descriptor. This leaves a gap in the payload buffer associated with the old descriptor. Since this gap is within a page, we can't just map it away in the linear user space mapping of the buffers, user space will see this gap and have to compensate, by copying, for example.

We obviously want to avoid gaps in the payload buffer, so setting up these descriptors, we need to make sure that the header buffer is big enough to hold headers for all the packets it takes to fill up the payload buffer. Now the packetization process isn't deterministic – in simple cases where the remote device is sampling using a clock based off of the bus clock domain, then, for example, a 48kHz audio signal can send 6 samples every cycle or maybe 3 packets with 8 samples and one empty packet consistently. But if the AD converter is driven using a separate clock, there is going to be clock skew, and suddenly there might be an extra empty packet. And the thing is, even without the clock skew problem, you don't know how the remote device is going to throttle the packets. All this to say that for a given payload size, there is no way to reliably know how many packets the remote device will use to transmit that payload.

A gap doesn't necessarily have to be a big problem as long as we know its position and size. I don't think it's a lot of overhead to skip a gap once in a while. Not having any would be better of course.

The ability to skip a gap will have to be implemented in (some) clients anyway, because it is not certain that a no-data packet won't contain payload. The current class driver for audio devices from Apple sends payload along with its no-data packets. I don't really know if this is according to spec, but I assume so (haven't got them at hand).

Re: In-tree version of new FireWire drivers available

In our application (FreeBoB), we know in advance what the size of a packet is going to be, as we only use blocking transmission. We also know how many (non no-data) packets we want to receive before being notified (some fraction of the audio buffer size). So we can predict the size of the payload buffer rather well. As the average rate of the samples in the packets is known (being the samplerate, say 48kHz), the average amount of no-data packets can be calculated. Then add some margin and we are OK.

I don't see an easy way to use double-buffering for non-blocking transmission, that will have to be handled by the normal method.

That's the stumbling block I've been looking at (I've been side tracked by a couple of unrelated tasks, but I'm now back on track). So the ideas I've been considering are

- Always allocate a page for headers and a page for the payload. This is a pretty simple solution that works as long as we're not streaming really small payloads compared to the header we slice off. So for example, mono 24kHz audio (I dunno, a dedicated subwoofer stream) would be an average of 3 quadlets payload against the 3 header quadlets (1 iso header quadlet + 2 iec61883 header quadlets). Of course, when streaming video with ~200 bytes payload, we're wasting most of a page of memory to receive 20 or so headers.

- Punt to user space. Ask user space to specify the minimum number of headers required to receive a certain payload. It's not unreasonable for user space to know this or at least be able to give a good estimate and add some margin. However, another problem with the dual buffer descriptors comes up here. If the payload crosses a page boundary, the kernel DMA logic needs to know how many headers to allocate for the first part and how many to allocate for the second part.

Wouldn't that be the 'maximum number of headers', because you want the payload to trigger the next descriptor switch?

What about having userspace specify the payload they want on a descriptor switch (hence interrupt), and the maximum number of headers they expect for this payload? For us it is all about the payload...

But maybe I'm just fuzzing over this issue. Since the header pages are only allocated while receiving, maybe the first idea is fine. And leaking what is essentially a OHCI dual buffer specific limitation to user space doesn't seem like a nice idea. So for now I'll try to get the first idea going and post an update as soon as I have something working.

I don't think that this is really an OHCI limitation but rather a side effect of the rather indeterministic nature of isochronous firewire transfer.

Re: In-tree version of new FireWire drivers available

Thanks for the elaborate answer!

Pieter

—

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>