

Re: [RFC v2 2/5] dmaengine: Add slave DMA interface

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-01/msg12645.html>

- *From:* David Brownell <david-b@xxxxxxxxxxxxx>
 - *Date:* Wed, 30 Jan 2008 02:52:49 -0800
-

On Wednesday 30 January 2008, Haavard Skinnemoen wrote:

Ok, I didn't bother to check the specs, as I think your main argument is that these options vary from controller to controller, so we need to make this extensible.

Not all options are specific to DMA slave transfers either, although most of them are probably more important in this context.

Right ...

Descriptor-based vs. register-based transfers sounds like something the DMA engine driver is free to decide on its own.

Not entirely. The current interface has "dma_async_tx_descriptor" wired pretty thoroughly into the call structure -- hard to avoid. (And where's the "dma_async_rx_descriptor", since that's only TX?? Asymmetry like that is usually not a healthy sign.) The engine is not free to avoid those descriptors ...

And consider that many DMA transfers can often be started (after cache synch operations) by writing less than half a dozen registers: source address, destination address, params, length, enable. Being wildly generous, let's call that a couple dozen instructions, including saving "what to do when it's done". The current framework requires several calls just to fill descriptors ... burning lots more than that many instructions even before getting around to the Real Work! (So I was getting at low DMA overheads there, more than any particular way to talk to the controller.)

Example: USB tends to use one packet per "frame" and have the DMA request signal mean "give me the next frame". It's sometimes been

Re: [RFC v2 2/5] dmaengine: Add slave DMA interface

very important to use the tuning options to avoid some on-chip race conditions for transfers that cross lots of internal busses and clock domains, and to have special handling for aborting transfers and handling "short RX" packets.

Is it enough to set these options on a per-channel basis, or do they have to be per-transfer?

Some depend on the buffer alignment and size, so "per-transfer" is the norm. Of course, if there aren't many channels, the various clients may need to recycle them a lot ... which means lots of setup anyway.

That particular hardware has enough of the "logical" channels that each driver gets its own; one level of arbitration involves assigning those to underlying "physical" channels.

I wonder whether a unified programming interface is the right way to approach peripheral DMA support, given such variability. The DMAC from Synopsys that you're working with has some of those options, but not all of them... and other DMA controllers have their own oddities.

Yes, but I still think it's worthwhile to have a common interface to common functionality. Drivers for hardware that does proper flow control won't need to mess with priority and arbitration settings anyway, although they could do it in order to tweak performance.

I wouldn't assume that systems have that much overcapacity on their critical I/O paths. I've certainly seen systems tune those busses down in speed ... you might describe the "why" as "tweaking battery performance", which wasn't at all an optional stage of the system development process.

So a plain "write this memory block to the TX register of this slave" interface will be useful in many cases.

That's a fair place to start. Although in my limited experience, drivers won't stay there. I have one particular headache in mind, where the licensed IP has been glued to at least four different DMA controllers. None of them act similarly -- sanely? -- enough to share much DMA interface code. Initiation has little quirks; termination has bigger ones; transfer aborts... sigh

Re: [RFC v2 2/5] dmaengine: Add slave DMA interface

Heck, you've seen similar stuff yourself with the MCI driver. AVR32 and AT91 have different DMA models. You chose to have them use different drivers...

For memcpy() acceleration, sure -- there shouldn't be much scope for differences. Source, destination, bytecount ... go! (Not that it's anywhere *near* that quick in the current interface.)

Well, I can imagine copying scatterlists may be useful too. Also, some controllers support "stride" (or "scatter-gather", as Synopsys calls it), which can be useful for framebuffer bitblt acceleration, for example.

The OMAP1 DMA controller supports that. There were also some framebuffer-specific DMA options. ;)

For peripheral DMA, maybe it should be a "core plus subclasses" approach so that platform drivers can make use hardware-specific knowledge (SOC-specific peripheral drivers using SOC-specific DMA), sharing core code for dma-memcpy() and DMA channel housekeeping.

I mostly agree, but I think providing basic DMA slave transfers only through extensions will cause maintenance nightmares for the drivers using it.

See above ... if you've got a driver that's got to cope with different DMA engines, those may be inescapable.

But I suppose we could have something along the lines of "core plus standard subclasses plus SOC-specific subclasses"...

That seems like one layer too many!

We already have something along those lines through the capabilities mask, taking care of the "standard subclasses" part. How about we add some kind of type ID to struct dma_device so that a driver can use container_of() to get at the extended bits if it recognizes the type?

Re: [RFC v2 2/5] dmaengine: Add slave DMA interface

That would seem to be needed if the interface isn't going to become a least-common-denominator approach -- or a kitchen-sink.

- Dave

Haavard

--

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

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

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