PXA/MMP - DMA Slave controller

Constraints

a) Transfers hot queuing A driver submitting a transfer and issuing it should be granted the transfer is queued even on a running DMA channel. This implies that the queuing doesn't wait for the previous transfer end, and that the descriptor chaining is not only done in the irq/tasklet code triggered by the end of the transfer. A transfer which is submitted and issued on a phy doesn't wait for a phy to stop and restart, but is submitted on a "running channel". The other drivers, especially mmp_pdma waited for the phy to stop before relaunching a new transfer.

b) All transfers having asked for confirmation should be signaled Any issued transfer with DMA_PREP_INTERRUPT should trigger a callback call. This implies that even if an irq/tasklet is triggered by end of tx1, but at the time of irq/dma tx2 is already finished, tx1->complete() and tx2->complete() should be called.

c) Channel running state A driver should be able to query if a channel is running or not. For the multimedia case, such as video capture, if a transfer is submitted and then a check of the DMA channel reports a "stopped channel", the transfer should not be issued until the next "start of frame interrupt", hence the need to know if a channel is in running or stopped state.

d) Bandwidth guarantee The PXA architecture has 4 levels of DMAs priorities : high, normal, low. The high priorities get twice as much bandwidth as the normal, which get twice as much as the low priorities. A driver should be able to request a priority, especially the real-time ones such as pxa_camera with (big) throughputs.

Design

a) Virtual channels Same concept as in sa11x0 driver, ie. a driver was assigned a "virtual channel" linked to the requestor line, and the physical DMA channel is assigned on the fly when the transfer is issued.

b. Transfer anatomy for a scatter-gather transfer

```
+------------+-----+---------------+----------------+-----------------+
| desc-sg[0] | ... | desc-sg[last] | status updater | finisher/linker |
+------------+-----+---------------+----------------+-----------------+
```

This structure is pointed by dma->sg_cpu. The descriptors are used as follows :

- desc-sg[i]: i-th descriptor, transferring the i-th sg element to the video buffer scatter gather
- status updater Transfers a single u32 to a well known dma coherent memory to leave a trace that this transfer is done. The "well known" is unique per physical channel, meaning that a read of this value will tell which is the last finished transfer at that point in time.
- finisher: has ddadr=DADDR_STOP, dcmd=ENDIRQEN
- linker: has ddadr= desc-sg[0] of next transfer, dcmd=0

c) Transfers hot-chaining Suppose the running chain is:

```
Buffer 1              Buffer 2
+---------+----+---+  +----+----+----+---+
| d0 | .. | dN | l |  | d0 | .. | dN | f |
+---------+----+-|-+  ^----+----+----+---+
                |    |
                +----+
```

After a call to dmaengine_submit(b3), the chain will look like:

```
Buffer 1              Buffer 2              Buffer 3
+---------+----+---+  +----+----+----+---+  +----+----+----+---+
| d0 | .. | dN | l |  | d0 | .. | dN | l |  | d0 | .. | dN | f |
+---------+----+-|-+  ^----+----+----+-|-+  ^----+----+----+---+
                |    |                 |    |
                +----+                 +----+
                                      new_link
```

If while new_link was created the DMA channel stopped, it is _not_ restarted. Hot-chaining doesn't break the assumption that dma_async_issue_pending() is to be used to ensure the transfer is actually started.

One exception to this rule :

- if Buffer1 and Buffer2 had all their addresses 8 bytes aligned
- and if Buffer3 has at least one address not 4 bytes aligned
- then hot-chaining cannot happen, as the channel must be stopped, the "align bit" must be set, and the channel restarted As a consequence, such a transfer tx_submit() will be queued on the submitted queue, and this specific case if the DMA is already running in aligned mode.

d) Transfers completion updater Each time a transfer is completed on a channel, an interrupt might be generated or not, up to the

client's request. But in each case, the last descriptor of a transfer, the "status updater", will write the latest transfer being completed into the physical channel's completion mark.

This will speed up residue calculation, for large transfers such as video buffers which hold around 6k descriptors or more. This also allows without any lock to find out what is the latest completed transfer in a running DMA chain.

e) Transfers completion, irq and tasklet When a transfer flagged as "DMA_PREP_INTERRUPT" is finished, the dma irq is raised. Upon this interrupt, a tasklet is scheduled for the physical channel.

The tasklet is responsible for :

- reading the physical channel last updater mark
- calling all the transfer callbacks of finished transfers, based on that mark, and each transfer flags.

If a transfer is completed while this handling is done, a dma irq will be raised, and the tasklet will be scheduled once again, having a new updater mark.

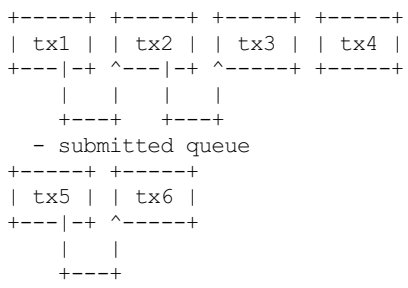
f) Residue Residue granularity will be descriptor based. The issued but not completed transfers will be scanned for all of their descriptors against the currently running descriptor.

g) Most complicated case of driver's tx queues The most tricky situation is when :

- there are not "acked" transfers (tx0)
- a driver submitted an aligned tx1, not chained
- a driver submitted an aligned tx2 => tx2 is cold chained to tx1
- a driver issued tx1+tx2 => channel is running in aligned mode
- a driver submitted an aligned tx3 => tx3 is hot-chained
- a driver submitted an unaligned tx4 => tx4 is put in submitted queue, not chained
- a driver issued tx4 => tx4 is put in issued queue, not chained
- a driver submitted an aligned tx5 => tx5 is put in submitted queue, not chained
- a driver submitted an aligned tx6 => tx6 is put in submitted queue, cold chained to tx5

This translates into (after tx4 is issued) :

- issued queue



- completed queue : empty
- allocated queue : tx0

It should be noted that after tx3 is completed, the channel is stopped, and restarted in "unaligned mode" to handle tx4.

Author: Robert Jarzmik <robert.jarzmik@free.fr>