### **Device operations**

Our dma\_device structure also requires a few function pointers in order to implement the actual logic, now that we described what operations we were able to perform.

The functions that we have to fill in there, and hence have to implement, obviously depend on the transaction types you reported as supported.

* device\_alloc\_chan\_resources
* device\_free\_chan\_resources
  + These functions will be called whenever a driver will call dma\_request\_channel or dma\_release\_channel for the first/last time on the channel associated to that driver.
  + They are in charge of allocating/freeing all the needed resources in order for that channel to be useful for your driver.
  + These functions can sleep.
* device\_prep\_dma\_\*
  + These functions are matching the capabilities you registered previously.
  + These functions all take the buffer or the scatterlist relevant for the transfer being prepared, and should create a hardware descriptor or a list of hardware descriptors from it
  + These functions can be called from an interrupt context
  + Any allocation you might do should be using the GFP\_NOWAIT flag, in order not to potentially sleep, but without depleting the emergency pool either.
  + Drivers should try to pre-allocate any memory they might need during the transfer setup at probe time to avoid putting to much pressure on the nowait allocator.
  + It should return a unique instance of the dma\_async\_tx\_descriptor structure, that further represents this particular transfer.
  + This structure can be initialized using the function dma\_async\_tx\_descriptor\_init.
  + You’ll also need to set two fields in this structure:
    - flags: TODO: Can it be modified by the driver itself, or should it be always the flags passed in the arguments
    - tx\_submit: A pointer to a function you have to implement, that is supposed to push the current transaction descriptor to a pending queue, waiting for issue\_pending to be called.
  + In this structure the function pointer callback\_result can be initialized in order for the submitter to be notified that a transaction has completed. In the earlier code the function pointer callback has been used. However it does not provide any status to the transaction and will be deprecated. The result structure defined as dmaengine\_result that is passed in to callback\_result has two fields:
    - result: This provides the transfer result defined by dmaengine\_tx\_result. Either success or some error condition.
    - residue: Provides the residue bytes of the transfer for those that support residue.
* device\_issue\_pending
  + Takes the first transaction descriptor in the pending queue, and starts the transfer. Whenever that transfer is done, it should move to the next transaction in the list.
  + This function can be called in an interrupt context
* device\_tx\_status
  + Should report the bytes left to go over on the given channel
  + Should only care about the transaction descriptor passed as argument, not the currently active one on a given channel
  + The tx\_state argument might be NULL
  + Should use dma\_set\_residue to report it
  + In the case of a cyclic transfer, it should only take into account the current period.
  + This function can be called in an interrupt context.
* device\_config
  + Reconfigures the channel with the configuration given as argument
  + This command should NOT perform synchronously, or on any currently queued transfers, but only on subsequent ones
  + In this case, the function will receive a dma\_slave\_config structure pointer as an argument, that will detail which configuration to use.
  + Even though that structure contains a direction field, this field is deprecated in favor of the direction argument given to the prep\_\* functions
  + This call is mandatory for slave operations only. This should NOT be set or expected to be set for memcpy operations. If a driver support both, it should use this call for slave operations only and not for memcpy ones.
* device\_pause
  + Pauses a transfer on the channel
  + This command should operate synchronously on the channel, pausing right away the work of the given channel
* device\_resume
  + Resumes a transfer on the channel
  + This command should operate synchronously on the channel, resuming right away the work of the given channel
* device\_terminate\_all
  + Aborts all the pending and ongoing transfers on the channel
  + For aborted transfers the complete callback should not be called
  + Can be called from atomic context or from within a complete callback of a descriptor. Must not sleep. Drivers must be able to handle this correctly.
  + Termination may be asynchronous. The driver does not have to wait until the currently active transfer has completely stopped. See device\_synchronize.
* device\_synchronize
  + Must synchronize the termination of a channel to the current context.
  + Must make sure that memory for previously submitted descriptors is no longer accessed by the DMA controller.
  + Must make sure that all complete callbacks for previously submitted descriptors have finished running and none are scheduled to run.
  + May sleep.

## **Misc notes**

(stuff that should be documented, but don’t really know where to put them)

dma\_run\_dependencies

* Should be called at the end of an async TX transfer, and can be ignored in the slave transfers case.
* Makes sure that dependent operations are run before marking it as complete.

dma\_cookie\_t

* it’s a DMA transaction ID that will increment over time.
* Not really relevant any more since the introduction of virt-dma that abstracts it away.

DMA\_CTRL\_ACK

* If clear, the descriptor cannot be reused by provider until the client acknowledges receipt, i.e. has has a chance to establish any dependency chains
* This can be acked by invoking async\_tx\_ack()
* If set, does not mean descriptor can be reused

DMA\_CTRL\_REUSE

* If set, the descriptor can be reused after being completed. It should not be freed by provider if this flag is set.
* The descriptor should be prepared for reuse by invoking dmaengine\_desc\_set\_reuse() which will set DMA\_CTRL\_REUSE.
* dmaengine\_desc\_set\_reuse() will succeed only when channel support reusable descriptor as exhibited by capabilities
* As a consequence, if a device driver wants to skip the dma\_map\_sg() and dma\_unmap\_sg() in between 2 transfers, because the DMA’d data wasn’t used, it can resubmit the transfer right after its completion.
* Descriptor can be freed in few ways
  + Clearing DMA\_CTRL\_REUSE by invoking dmaengine\_desc\_clear\_reuse() and submitting for last txn
  + Explicitly invoking dmaengine\_desc\_free(), this can succeed only when DMA\_CTRL\_REUSE is already set
  + Terminating the channel
* DMA\_PREP\_CMD
  + If set, the client driver tells DMA controller that passed data in DMA API is command data.
  + Interpretation of command data is DMA controller specific. It can be used for issuing commands to other peripherals/register reads/register writes for which the descriptor should be in different format from normal data descriptors.