Linux I2C and DMA

Given that I2C is a low-speed bus, over which the majority of messages transferred are small, it is not considered a prime user of DMA access. At this time of writing, only 10% of I2C bus master drivers have DMA support implemented. And the vast majority of transactions are so small that setting up DMA for it will likely add more overhead than a plain PIO transfer.

Therefore, it is *not* mandatory that the buffer of an I2C message is DMA safe. It does not seem reasonable to apply additional burdens when the feature is so rarely used. However, it is recommended to use a DMA-safe buffer if your message size is likely applicable for DMA. Most drivers have this threshold around 8 bytes (as of today, this is mostly an educated guess, however). For any message of 16 byte or larger, it is probably a really good idea. Please note that other subsystems you use might add requirements. E.g., if your I2C bus master driver is using USB as a bridge, then you need to have DMA safe buffers always, because USB requires it.

Clients

For clients, if you use a DMA safe buffer in i2c_msg, set the I2C_M_DMA_SAFE flag with it. Then, the I2C core and drivers know they can safely operate DMA on it. Note that using this flag is optional. I2C host drivers which are not updated to use this flag will work like before. And like before, they risk using an unsafe DMA buffer. To improve this situation, using I2C_M_DMA_SAFE in more and more clients and host drivers is the planned way forward. Note also that setting this flag makes only sense in kernel space. User space data is copied into kernel space anyhow. The I2C core makes sure the destination buffers in kernel space are always DMA capable. Also, when the core emulates SMBus transactions via I2C, the buffers for block transfers are DMA safe. Users of i2c_master_send() and i2c_master_recv() functions can now use DMA safe variants (i2c_master_send_dmasafe() and i2c_master_recv_dmasafe()) once they know their buffers are DMA safe. Users of i2c_transfer() must set the I2C_M_DMA_SAFE flag manually.

Masters

Bus master drivers wishing to implement safe DMA can use helper functions from the I2C core. One gives you a DMA-safe buffer for a given i2c_msg as long as a certain threshold is met:

```
dma_buf = i2c_get_dma_safe_msg_buf(msg, threshold_in_byte);
```

If a buffer is returned, it is either msg->buf for the I2C_M_DMA_SAFE case or a bounce buffer. But you don't need to care about that detail, just use the returned buffer. If NULL is returned, the threshold was not met or a bounce buffer could not be allocated. Fall back to PIO in that case.

In any case, a buffer obtained from above needs to be released. Another helper function ensures a potentially used bounce buffer is freed:

```
i2c_put_dma_safe_msg_buf(dma_buf, msg, xferred);
```

The last argument 'xferred' controls if the buffer is synced back to the message or not. No syncing is needed in cases setting up DMA had an error and there was no data transferred.

The bounce buffer handling from the core is generic and simple. It will always allocate a new bounce buffer. If you want a more sophisticated handling (e.g. reusing pre-allocated buffers), you are free to implement your own.

Please also check the in-kernel documentation for details. The i2c-sh_mobile driver can be used as a reference example how to use the above helpers.

Final note: If you plan to use DMA with I2C (or with anything else, actually) make sure you have CONFIG_DMA_API_DEBUG enabled during development. It can help you find various issues which can be complex to debug otherwise.