I2C muxes and complex topologies

There are a couple of reasons for building more complex I2C topologies than a straight-forward I2C bus with one adapter and one or more devices.

1. A mux may be needed on the bus to prevent address collisions.
2. The bus may be accessible from some external bus master, and arbitration may be needed to determine if it is ok to access the bus.
3. A device (particularly RF tuners) may want to avoid the digital noise from the I2C bus, at least most of the time, and sits behind a gate that has to be operated before the device can be accessed.

Etc

These constructs are represented as I2C adapter trees by Linux, where each adapter has a parent adapter (except the root adapter) and zero or more child adapters. The root adapter is the actual adapter that issues I2C transfers, and all adapters with a parent are part of an "i2c-mux" object (quoted, since it can also be an arbitrator or a gate).

Depending of the particular mux driver, something happens when there is an I2C transfer on one of its child adapters. The mux driver can obviously operate a mux, but it can also do arbitration with an external bus master or open a gate. The mux driver has two operations for this, select and deselect. select is called before the transfer and (the optional) deselect is called after the transfer.

Locking

There are two variants of locking available to I2C muxes, they can be mux-locked or parent-locked muxes. As is evident from below, it can be useful to know if a mux is mux-locked or if it is parent-locked. The following list was correct at the time of writing:

In drivers/i2c/muxes/:

i2c-arb-gpio-challenge	Parent-locked
i2c-mux-gpio	Normally parent-locked, mux-locked iff all involved gpio pins are controlled by the same I2C root adapter that they mux.
i2c-mux-gpmux	Normally parent-locked, mux-locked iff specified in device-tree.
i2c-mux-ltc4306	Mux-locked
i2c-mux-mlxcpld	Parent-locked
i2c-mux-pca9541	Parent-locked
i2c-mux-pca954x	Parent-locked
i2c-mux-pinctrl	Normally parent-locked, mux-locked iff all involved pinctrl devices are controlled by the same I2C root adapter that they mux.
i2c-mux-reg	Parent-locked

In drivers/iio/:

gyro/mpu3050	Mux-locked
imu/inv_mpu6050/	Mux-locked

In drivers/media/:

dvb-frontends/lgdt3306a	Mux-locked
dvb-frontends/m88ds3103	Parent-locked
dvb-frontends/rtl2830	Parent-locked
dvb-frontends/rtl2832	Mux-locked
dvb-frontends/si2168	Mux-locked
usb/cx231xx/	Parent-locked

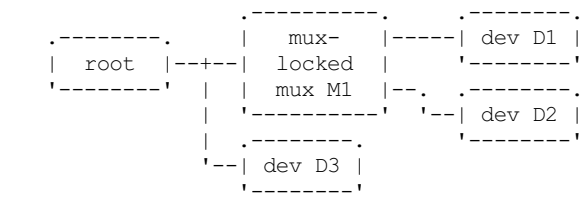
Mux-locked muxes

Mux-locked muxes does not lock the entire parent adapter during the full select-transfer-deselect transaction, only the muxes on the parent adapter are locked. Mux-locked muxes are mostly interesting if the select and/or deselect operations must use I2C transfers to complete their tasks. Since the parent adapter is not fully locked during the full transaction, unrelated I2C transfers may interleave the different stages of the transaction. This has the benefit that the mux driver may be easier and cleaner to implement, but it has some caveats.

ML1.	If you build a topology with a mux-locked mux being the parent of a parent-locked mux, this might break the expectation from the parent-locked mux that the root adapter is locked during the transaction.

ML2.	It is not safe to build arbitrary topologies with two (or more) mux-locked muxes that are not siblings, when there are address collisions between the devices on the child adapters of these non-sibling muxes. I.e. the select-transfer-deselect transaction targeting e.g. device address 0x42 behind mux-one may be interleaved with a similar operation targeting device address 0x42 behind mux-two. The intension with such a topology would in this hypothetical example be that mux-one and mux-two should not be selected simultaneously, but mux-locked muxes do not guarantee that in all topologies.
ML3.	A mux-locked mux cannot be used by a driver for auto-closing gates/muxes, i.e. something that closes automatically after a given number (one, in most cases) of I2C transfers. Unrelated I2C transfers may creep in and close prematurely.
ML4.	If any non-I2C operation in the mux driver changes the I2C mux state, the driver has to lock the root adapter during that operation. Otherwise garbage may appear on the bus as seen from devices behind the mux, when an unrelated I2C transfer is in flight during the non-I2C mux-changing operation.

Mux-locked Example



When there is an access to D1, this happens:

1. Someone issues an I2C transfer to D1.
2. M1 locks muxes on its parent (the root adapter in this case).
3. M1 calls ->select to ready the mux.
4. M1 (presumably) does some I2C transfers as part of its select. These transfers are normal I2C transfers that locks the parent adapter.
5. M1 feeds the I2C transfer from step 1 to its parent adapter as a normal I2C transfer that locks the parent adapter.
6. M1 calls ->deselect, if it has one.
7. Same rules as in step 4, but for ->deselect.
8. M1 unlocks muxes on its parent.

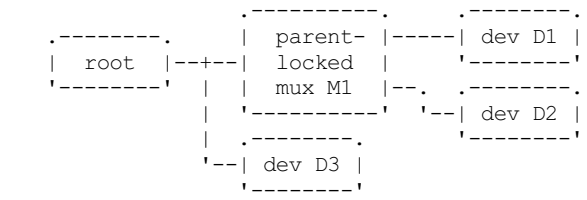
This means that accesses to D2 are lockout out for the full duration of the entire operation. But accesses to D3 are possibly interleaved at any point.

Parent-locked muxes

Parent-locked muxes lock the parent adapter during the full select- transfer-deselect transaction. The implication is that the mux driver has to ensure that any and all I2C transfers through that parent adapter during the transaction are unlocked I2C transfers (using e.g. __i2c_transfer), or a deadlock will follow. There are a couple of caveats.

PL1.	If you build a topology with a parent-locked mux being the child of another mux, this might break a possible assumption from the child mux that the root adapter is unused between its select op and the actual transfer (e.g. if the child mux is auto-closing and the parent mux issues I2C transfers as part of its select). This is especially the case if the parent mux is mux-locked, but it may also happen if the parent mux is parent-locked.
PL2.	If select/deselect calls out to other subsystems such as gpio, pinctrl, regmap or iio, it is essential that any I2C transfers caused by these subsystems are unlocked. This can be convoluted to accomplish, maybe even impossible if an acceptably clean solution is sought.

Parent-locked Example



When there is an access to D1, this happens:

1. Someone issues an I2C transfer to D1.
2. M1 locks muxes on its parent (the root adapter in this case).
3. M1 locks its parent adapter.
4. M1 calls ->select to ready the mux.

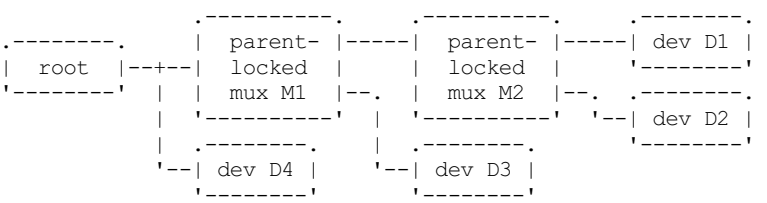
5. If M1 does any I2C transfers (on this root adapter) as part of its select, those transfers must be unlocked I2C transfers so that they do not deadlock the root adapter.
6. M1 feeds the I2C transfer from step 1 to the root adapter as an unlocked I2C transfer, so that it does not deadlock the parent adapter.
7. M1 calls ->deselect, if it has one.
8. Same rules as in step 5, but for ->deselect.
9. M1 unlocks its parent adapter.
10. M1 unlocks muxes on its parent.

This means that accesses to both D2 and D3 are locked out for the full duration of the entire operation.

Complex Examples

Parent-locked mux as parent of parent-locked mux

This is a useful topology, but it can be bad:



When any device is accessed, all other devices are locked out for the full duration of the operation (both muxes lock their parent, and specifically when M2 requests its parent to lock, M1 passes the buck to the root adapter).

This topology is bad if M2 is an auto-closing mux and M1->select issues any unlocked I2C transfers on the root adapter that may leak through and be seen by the M2 adapter, thus closing M2 prematurely.

Mux-locked mux as parent of mux-locked mux

This is a good topology:

When device D1 is accessed, accesses to D2 are locked out for the full duration of the operation (muxes on the top child adapter of M1 are locked). But accesses to D3 and D4 are possibly interleaved at any point. Accesses to D3 locks out D1 and D2, but accesses to D4 are still possibly interleaved.

Mux-locked mux as parent of parent-locked mux

This is probably a bad topology:

When device D1 is accessed, accesses to D2 and D3 are locked out for the full duration of the operation (M1 locks child muxes on the root adapter). But accesses to D4 are possibly interleaved at any point.

This kind of topology is generally not suitable and should probably be avoided. The reason is that M2 probably assumes that there will be no I2C transfers during its calls to ->select and ->deselect, and if there are, any such transfers might appear on the slave side of M2 as partial I2C transfers, i.e. garbage or worse. This might cause device lockups and/or other problems.

The topology is especially troublesome if M2 is an auto-closing mux. In that case, any interleaved accesses to D4 might close M2 prematurely, as might any I2C transfers part of M1->select.

But if M2 is not making the above stated assumption, and if M2 is not auto-closing, the topology is fine.

Parent-locked mux as parent of mux-locked mux

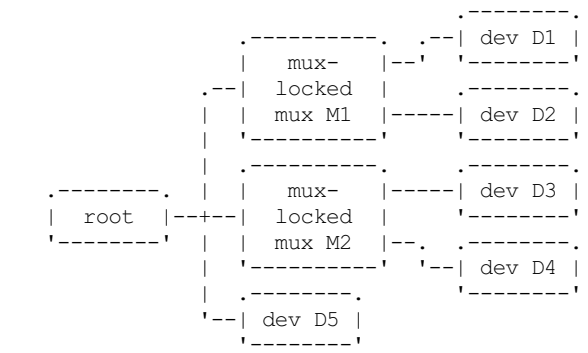
This is a good topology:

When D1 is accessed, accesses to D2 are locked out for the full duration of the operation (muxes on the top child adapter of M1 are locked). Accesses to D3 and D4 are possibly interleaved at any point, just as is expected for mux-locked muxes.

When D3 or D4 are accessed, everything else is locked out. For D3 accesses, M1 locks the root adapter. For D4 accesses, the root adapter is locked directly.

Two mux-locked sibling muxes

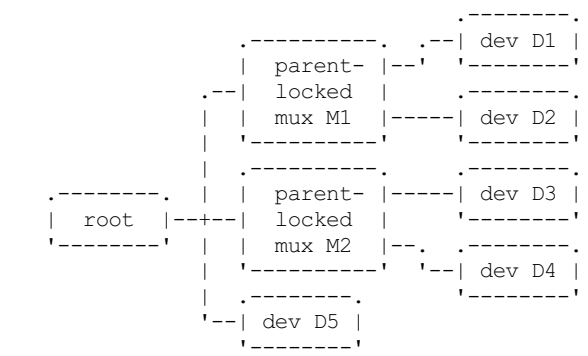
This is a good topology:



When D1 is accessed, accesses to D2, D3 and D4 are locked out. But accesses to D5 may be interleaved at any time.

Two parent-locked sibling muxes

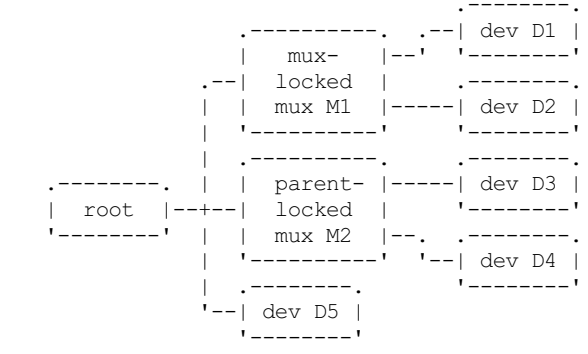
This is a good topology:



When any device is accessed, accesses to all other devices are locked out.

Mux-locked and parent-locked sibling muxes

This is a good topology:



When D1 or D2 are accessed, accesses to D3 and D4 are locked out while accesses to D5 may interleave. When D3 or D4 are accessed, accesses to all other devices are locked out.