Need for Bidirectional Event Handlers in Auxiliary Bus

Type: BoF / Discussion (45–60 min) **Suggested track:** Driver APIs / Device Model

---

Abstract

The auxiliary bus gives us a clean way to split a single PCI device into multiple cooperating drivers, but its communication model is limited: the parent (aux_device creator) exposes data to the child (aux_driver), and the child consumes them. There is no standardized mechanism for either side to *notify* the other of events (asynchronous or synchronous): link change, reset, FLR, capability change, config update, teardown intent, or specific kernel API calls targeted at the child but requiring parent resources to complete.

Example pain points:

- DCBNL API configuration requests come into the netdev owner but require configuration of PCI driver owned HW resources.
- Netshaper requests a rate limit set on an entity through netdev anchored request, but configuration requires access to a PCI driver owned channel to the FW.
- Netdev owner enables queues, which require configuration of the Scheduling Tree, and the tree must be configured through PCI owned resources.

Today every subsystem reinvents this:

- **mlx5** — per-driver notifier chains
- **ice / irdma** — exported symbols which create strange dependencies.
- **cxl, dsa** — ad-hoc ops structs.
- **idpf** — (proposed) shared ops + event delivery

This BoF proposes a *bidirectional event-handler* pattern usable by any auxiliary_bus consumer, presents our IDPF-based implementation as an illustration, and asks the room whether this should become the common convention.

Background (5 min)

- auxiliary_bus refresher: device/driver lifecycle, match by name, probe/remove, who owns what.
- Why aux_bus instead of MFD, platform_bus, or a custom bus.

The Problem (5 min)

A modular NIC / IPU stack needs **both** directions:

Parent (PCI owner) → Child (netdev owner)

- link up/down
- reset / FLR pending
- capability or resource change
- "prepare to detach"
- FW-generated event

Child (netdev owner) → Parent (PCI owner)

- request reset
- request MAC / VLAN / queue config
- ethtool / netshaper operation needing HW
- DCBNL config change request

Having bidirectional event handlers avoids a large and growing set of callback ops pointers on both sides.

Our Proposal (10 min)

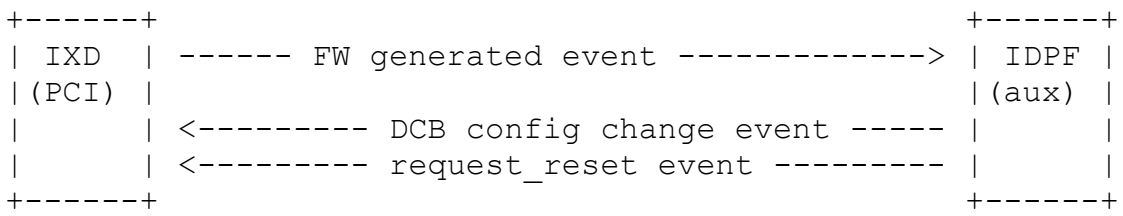
A small, symmetric event-handler contract carried in the shared aux_dev info struct (e.g. idpf_eth_aux_dev_info).

Key properties:

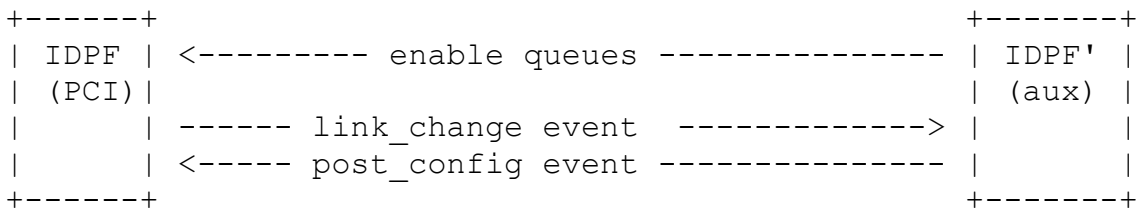
- Events are an **enum**.
- Payloads are members of a **union** controlled by event type.
- Unknown events return -EOPNOTSUPP rather than crashing.
- Lifetime is anchored to aux_device get/put.
- Calls to the event handler require device_lock and NULL checks.

Walkthrough: How It Plays Out in IXD/IDPF (10 min)

Model A — Foundational NIC



Model B — IPU



Same contract, two very different parents. That is the point.

Open Questions for the Room (20–25 min — the actual BoF)

1. **Scope**
    - Should bidirectional events live in core auxiliary_bus, or stay a per-subsystem convention?
2. **Existing mechanisms — do we actually need a new one?**
    - notifier_chain / atomic_notifier / blocking_notifier
    - devlink notifications and devlink ops
    - netlink (genl) as a cross-driver event bus
    - SRCU-based callback lists (mlx5 style)

    Which of these are we reinventing, and why aren't they enough?
3. **Locking**
    - Are events allowed to sleep? Re-enter the bus? Take rtnl?
4. **Discoverability & ABI**
    - How does each side learn which events the other supports?
5. **Naming, where it lives.**
    - drivers/base/auxiliary*.c additions?
    - include/linux/auxiliary_bus.h vs a new header?

Desired Outcomes

- Consensus (or at least direction) on whether this belongs in core.
- A short list of must-have semantics (lifetime, locking, versioning) any implementation must satisfy.
- Identification of other subsystems (mlx5, cxl, irdma, dsa, gpu) that would adopt or co-design this.
- A volunteer reviewer pool for the eventual RFC.

Pre-reads (to be provided)

- Link to our RFC patch / branch.