Agenda

- Overview of On-Chip interconnects
  - Challenges of SoC architecture
  - Evolution of On-Chip interconnects
  - The Network-On-Chip concept

- Interconnects and the Linux kernel
  - Requirements
  - Existing infrastructure

- Solutions and future work
Challenges of SoC Architecture

- High demand for integrating more and more new features (IP cores)
- Multiple processors (CPUs, dedicated GPU, video decoder, graphics cores, hardware accelerators, DSP, modem, camera, wifi, bluetooth, GPS etc.)
- Many components talking to each other
- High speed, fast transfers, better throughput etc.
- Multiple sources of traffic
- Predictability (many interrupts, concurrent transfers, DDR utilization)
Evolution of On-Chip interconnects

- Roles of the interconnects
- Buses and crossbars
- More complex hardware
- Small footprint
- Verification becomes more difficult
Topologies

Bus

Crossbar
Topologies

Fully connected point to point network
The Network-On-Chip concept
NoC approach benefits

- Scalability
- Packet communication
- Shared resources (links, memory)
- Priority and QoS
- Load balancing
- Power efficiency (short wires)
Network-On-Chip and Linux

- We want to control the NoCs and use of all benefits
- But how to do this in Linux...this has been not fully addressed yet
- Out of the tree implementations
- How does this fit in the Linux kernel?
An example use case

- Device X wants to reach device Y over a few NoCs
- On what port is device X connected
- What are the endpoints which device X can access
- Find the route to the destination endpoint
- Each node aggregates the requests and set QoS in hardware
An example topology
An example topology
What do we need to solve this?

- Topology description of how the devices and buses are interconnected
- API for setting constraints on devices (source and destination pair)
- Driver for the vendor specific hardware
- QoS update triggers
Use existing Linux frameworks and infrastructure 1/2

- The driver model
- Device-tree
- Consumer/provider based API
- Power management
- Operating Performance Points
- Devfreq
Use existing Linux frameworks and infrastructure

2/2

- Dynamic Voltage and Frequency Scaling
- Runtime PM
- Generic PM domains
- PMQoS
- Issues - layering and abstraction
Consumer-provider model

- Already used in many kernel subsystems
- Model the interconnects and buses as performance providers
- The provider implementation is vendor-specific and should provide the following callbacks as a minimum:
  - `init()` - for initializing the hardware
  - `update()` - for updating/setting the constraints
- The consumers use get/put API functions
- Consumers set constraints on the path to an endpoint.
- The vendor-specific code updates all the nodes on the path
Device-tree

- Describe interconnects between devices
- Call the NoC hardware a performance provider
- Create a DT node for the NoC hardware controller

```c
noc:noc@0120000 {
    compatible = "vendor,noc";
    reg = <0x120000 0x10000>;
    #performance-cells = <1>;
}
```
Device-tree

- Call the consumer device a performance consumer
- How and where the device is connected
- Describe endpoints, flow direction, priorities etc.
- Use a get() API to claim the link resource
- Add link metrics?

```c
consumer: device@0340000 {
    ...
    performance-port = <&noc 4>; /* connected to port 4 on NoC */
    performance-names = "usb", "display"; /* give a name to each link */
    performance-links = <&usb>, <&display>; /* phandles to device nodes */
}
```
Per device PMQoS

- Already exists in the kernel but it is still not widely used
- Used to store constraints per each device.
- Add constraints based on source and destination device pair
- Extend to support more than just min/max/sum
Runtime PM

- Currently power management is idle-based
- The trigger is runtime_pm_put()
- Add support for active-based power management
- Use runtime_pm_get() as a trigger for updating the bus/device performance
Generic PM domains

- Allow grouping of devices with similar characteristics
- Currently used for hardware power domains
- Support governors
- Extend genpd to allow a device to be in multiple PM domains
The Solution?

- Performance framework or interconnect framework?
- Consumer-provider based model
- Use device-tree to describe where a device is connected and claim the “links” as resources
- “Functional dependencies between devices” patchset
- Extend PMQoS
- Use runtime_pm_get() as trigger
- Use Generic PMDomains for grouping devices?
- Involve more people from the community and from different vendors
Future work

- The “Functional dependencies between devices” patchset by Rafael Wysocki is planned for 4.10

- Extend PMQoS with source and destination device pairs?

- Add support for devices in multiple PM domains?


- Reminder: There will be a session for discussions after lunch!
Thank You