An Introduction to DisplayPort MST

Lyude Paul

Who am I?

- Software engineer at Red Hat
- Primarily work on graphics drivers, also have experience with input and a number of other areas
- Also a resident MST expert :)

What is DisplayPort MST?

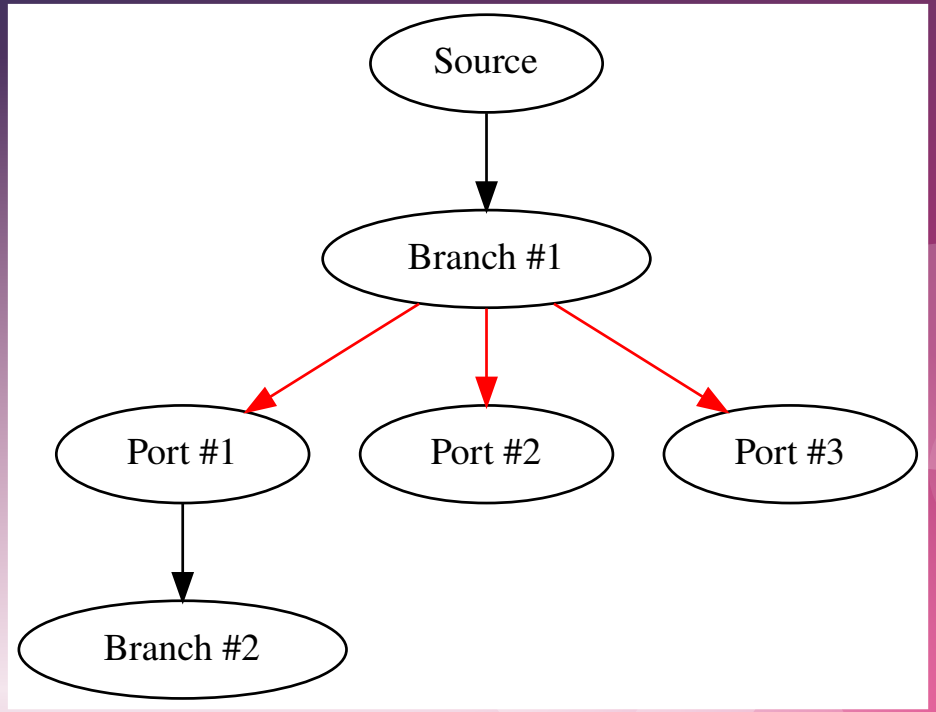
- A protocol for DisplayPort allowing the daisy-chaining of multiple displays on a single link
- Very common in laptop docks, some monitors use it for daisy-chaining
- Supported as of DisplayPort 1.2
- Supported in amdgpu, nouveau, and i915

Why make a presentation on this?

- There’s been a lot of reorganization of DP MST code to rely much more heavily on the atomic state
- Making people aware of the helpers
- Sometimes DP MST be quite rude, and is probably one of the top causes of display issues

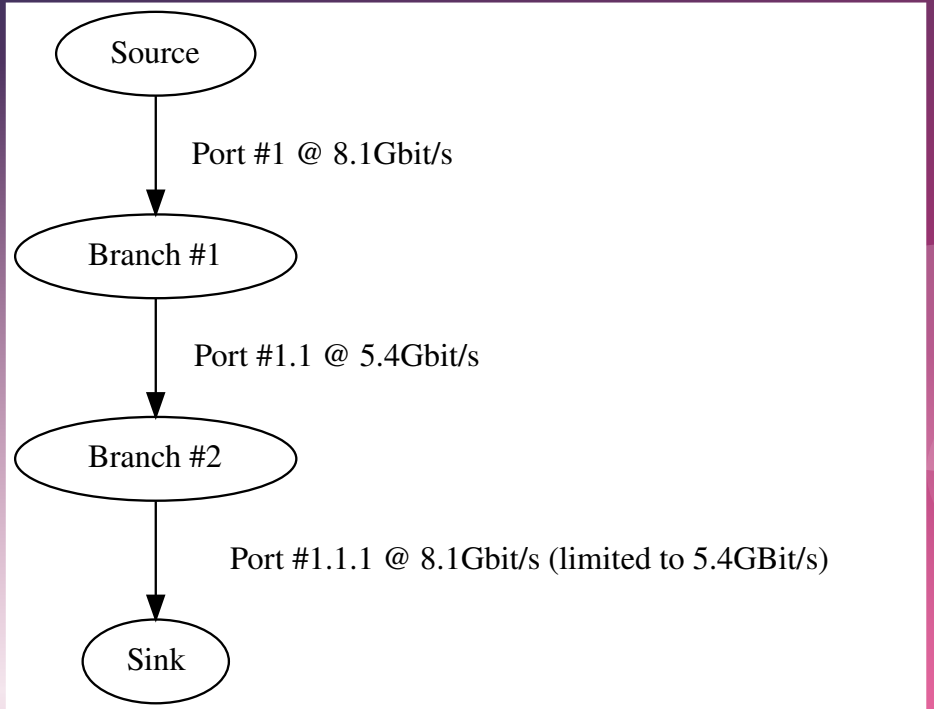
How does it work?

- Devices which provide ports on a topology are known as branches, and connectors are referred to as ports
- Ports on a branch share bandwidth
- Each active DP stream is referred to as a virtual channel, or VC
- Timeslots represent slices of the available bandwidth on a branch's immediate ports
- 63 or 64 Timeslots (8B/10B encoding)
- Each VC's PBN (payload bandwidth number) is rounded up to the closest equivalent in TUs



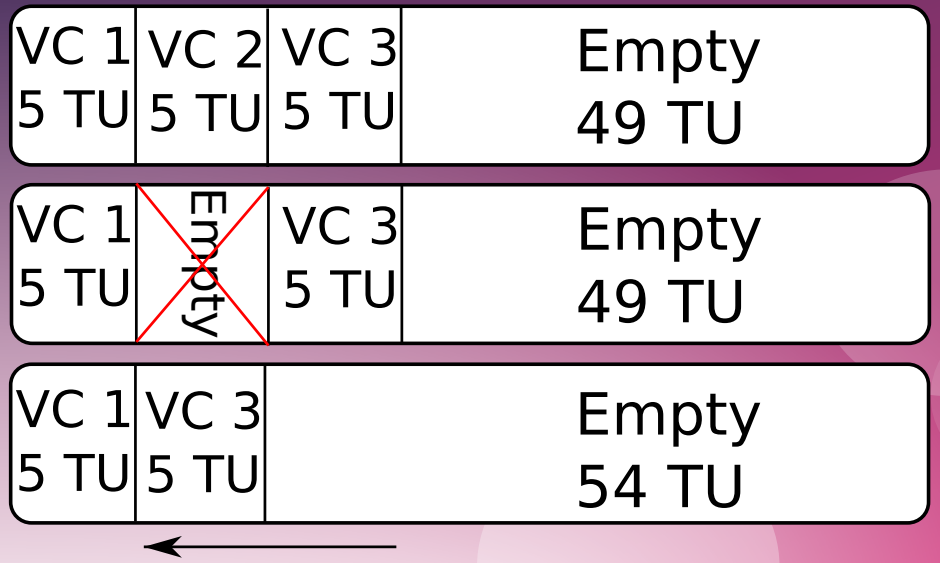
How does it work (cont.)

- Topologies can have multiple nested branches
- TUs are only for top branch though
- To deal with this, each port has a max PBN
- The source is expected to ensure the resulting PBN requirement for a payload on said port never exceeds this



Allocation/de-allocation

- Payload allocations are *always* contiguous, e.g. no holes
- Source specifies starting time slot and time slots/pbn allocated
- The sink/branch is expected to move payloads around automatically to fill any holes that appear as the result of releasing a payload
- Source is expected to keep track of this without reading back the payload table



Communication

- Basic communication happens via “sideband” communications
- Provides interface over DPCD similar to i2c over AUX, but is message oriented instead of register oriented
- Messages can be broadcast, or targeted at a specific RAD (relative address)
  - Each branch reads the RAD and routes it to the next closest branch on the path
  - It’s basically just a game of telephone
- Raw DPCD accesses on branches can be done via REMOTE_DPCD_READ/WRITE commands
- Otherwise, low level DPCD accesses between branches are handled by the branches themselves

Communication (cont.)

- Hotplugs, HDCP notifications and others are also received via sideband messages we receive from branches
- Interrupts are provided for these messages using short HPD IRQs

Troubleshooting

- Every branch has firmware running, and it very often breaks
- Number of reasons for this happening:
  - Parts of the MST spec have been implemented incorrectly on sinks, for example: interleaved sideband messaging
  - Many times these failures are not explicit and don’t provide us with error codes. For example, interleaved sideband messaging will partly work on many hubs but will still result in spontaneous timeouts
  - Sometimes things don’t even break cleanly, they just take way longer then they should!

Troubleshooting tips

- Even as problematic as MST hubs are, usually we're the culprit
- Or the cable. Good lord, the cables.
  - I'm serious here. This causes SO many weird problems
  - 2 cables is not enough, 3 is OK
    - Do not be like me and assume "oh, second cable is broken too so it can't be that". You'll regret it a week later.
  - Or just use USB-C, which does not have this problem
- Read the spec at least 10 million times because specs are hard and reading them is hard and we probably misread something

Troubleshooting

- Timeouts are usually:
  - The driver needing to be more async (e.g. handling sideband tx + rx simultaneously)
  - A not-fatal error occurring somewhere down the topology (for instance, a lower level branch failing link training somehow)
  - Just delay from having too many nested branch devices

State of affairs

- DRM has a set of MST related helpers for handling most of the hard stuff:
  - Topology probing
  - Sideband messaging
  - Bandwidth limitations
  - Etc.

State of affairs

- Certain features still missing:
  - DSC support – some helpers written, but we need more atomic state helpers for determining optimal DSC settings
  - Most of the DSC helpers can be pulled out of amdgpu code
  - Helpers for fallback retraining/bpp optimization
  - More comprehensive error handling
  - Testing testing testing
    - We have no tests
      - Please save us chamelium…
  - (we could probably write unit tests though)

Special thanks

- Wayne Lin for their work on DSC support, bug fixing and helping out with reviewing my work!
- VESA for being incredibly cool and helpful!
  - (X.org members can request access to VESA specs)
  - Everyone, say "*Thanks VESA*"
  - Also say "*Thanks Bill*"