



How to add ACPI support for RISC-V Trace?

Sunil V L

sunilvl@oss.qualcomm.com

Anup Patel

anup.patel@oss.qualcomm.com

Qualcomm, Inc

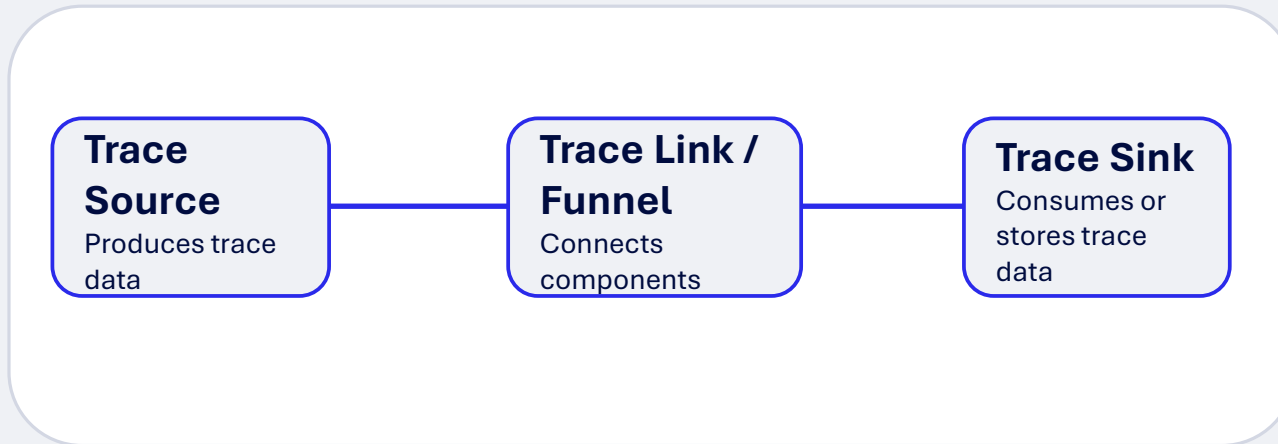
Linux Plumbers Conference 2026

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries.
Qualcomm patents are licensed by Qualcomm Incorporated.



Trace as a Device Graph in DT

Device Tree describes trace as a graph of components and connections—not merely a list of devices.



- Nodes represent trace components.
- Ports and endpoints describe the connection points.
- Remote endpoints identify the other side of each connection.

```
// Example 1 (Per-hart encoder and ramsink components):

trace@c000000 {
    compatible = "qemu,trace-component", "riscv,trace-component";
    reg = <0xc0000000 0x1000>;
    cpus = <&CPU0>;

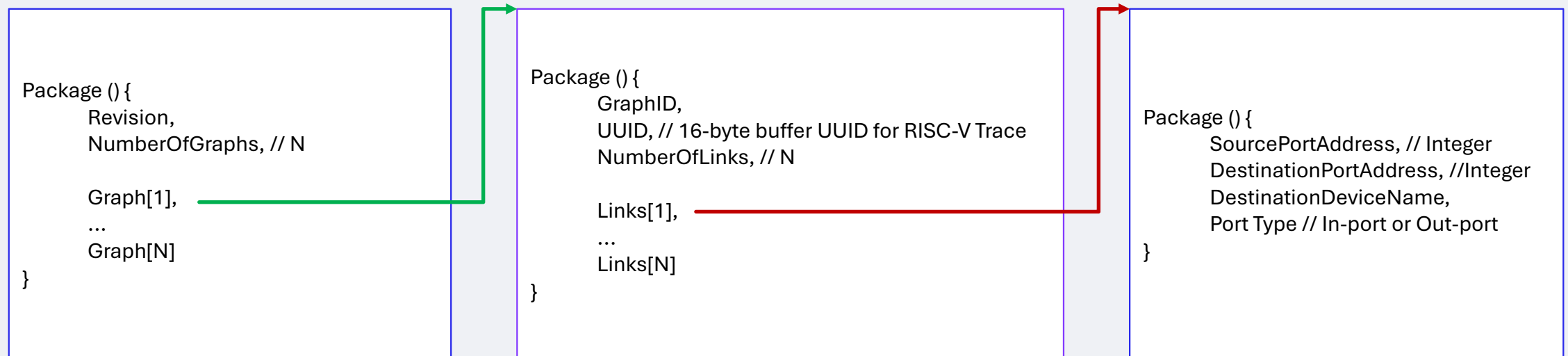
    out-ports {
        port {
            CPU0_ENCODER_OUTPUT: endpoint {
                remote-endpoint = <&CPU0_RAMSink_INPUT>;
            };
        };
    };
};

trace@c001000 {
    compatible = "qemu,trace-component", "riscv,trace-component";
    reg = <0xc0010000 0x1000>;
    cpus = <&CPU0>;

    in-ports {
        port {
            CPU0_RAMSink_INPUT: endpoint {
            };
        };
    };
};
```

Representing the Trace Graph in ACPI

- Trace components should be represented using a device graph model to clearly capture connectivity and relationships.
- The DSD Guide defines the Device Graph UUID (Section 3.4) specifically for describing such component interconnections.
- This approach is already adopted by other architectures. BRS 1.1 proposes the same for RISC-V.



What is missing for ACPI Trace graph in Linux?

The Core Issue

Linux already provides graph-traversal APIs for **Device Tree**:

The OF framework understands graph concepts such as:

- Ports
- Endpoints
- Remote connections
- Links and funnels

The architecture-specific trace driver can therefore use common OF APIs to discover and traverse the trace topology.

However, for **ACPI**:

- The trace graph described using the **DSD graph specification** is not natively supported by the Linux ACPI framework.
- The architecture-specific trace driver must parse and traverse the ACPI graph itself.
- This results in separate graph-discovery implementations for DT and ACPI.

Ref: `drivers/hwtracing/coresight/coresight-platform.c`

Linux Implementation for RISC-V : Option 1: Driver-local Graph Parsing

Approach:

Follow the existing CoreSight-style implementation by parsing the ACPI Device Graph UUID within the architecture-specific trace drivers.

Advantages:

- Reuses an established implementation model.
- Keeps changes localized to the architecture-specific drivers.
- Requires no changes to the common code.

Trade-offs:

- Duplicates graph parsing logic across drivers.
- Maintains separate ACPI and Device Tree code paths.
- Increases conditional code and long-term maintenance overhead.

```
if (is_of_node(fwnode))
    ret = of_get_coresight_platform_data(dev, pdata);
else if (is_acpi_device_node(fwnode))
    ret = acpi_get_coresight_platform_data(dev, pdata);
```

Linux Implementation for RISC-V : Option 2: Enhance fwnode framework

Approach:

Current ACPI fwnode graph interfaces do not support the Device Graph UUID, which has led other architectures to adopt architecture-specific implementations (Option 1). This proposal extends the common fwnode framework to provide native support for ACPI device graphs.

1. Parse the ACPI _DSD Device Graph and dynamically create graph nodes equivalent to DT nodes such as port, endpoint, and remote-endpoint.
2. Extend the ACPI fwnode backend so existing fwnode graph APIs can traverse ACPI device graphs transparently.
3. Enable drivers using the common fwnode graph interfaces to operate with both DT and ACPI with minimal or no driver-specific changes.

```
-    parent = of_get_child_by_name(dev_of_node(pdata->dev), "out-ports");  
+    parent = fwnode_get_named_child_node(dev_fwnode(pdata->dev), "out-ports");  
  
    if (!parent)  
        return 0;  
  
-    pdata->nr_outconns = of_graph_get_endpoint_count(parent);  
+    pdata->nr_outconns = fwnode_graph_get_endpoint_count(parent, 0);
```

Linux Implementation for RISC-V: Option 2: Enhance fwnode framework

Advantages:

- **Maximizes code reuse** by leveraging the existing fwnode graph infrastructure shared across DT and ACPI.
- **Reduces architecture-specific implementations**, avoiding duplicate graph parsing logic in multiple subsystems.
- **Keeps driver code largely the same**, allowing drivers to rely on common fwnode APIs rather than DT-specific or ACPI-specific interfaces.

Trade-offs:

- Requires dynamic creation and maintenance of DT-like graph nodes from ACPI _DSD Device Graph data.
- Introduces modifications in the common ACPI property/fwnode infrastructure.
- DT schema for graphs has not standardized the direction of the ports. Is that a concern given OF framework already supports “in-ports” / “out-ports”?
- Has a broader implementation footprint than Option 1 and requires additional review across ACPI and fwnode maintainers.

References

RFC patch series for Option 2: [\[RFC PATCH 0/5\] hwtracing: rvtrace: Add ACPI support](#)

DSD guide: <https://github.com/UEFI/DSD-Guide/blob/main/dsd-guide.pdf>

BRS 1.1: <https://github.com/riscv-non-isa/riscv-brs/releases/download/v1.0.6/brs-v1.0.6-20260928.pdf>

Discussion and Conclusion on Options

Thank you

Nothing in these materials is an offer to sell any of the components or devices referenced herein.

© Qualcomm Technologies, Inc. and/or its affiliated companies. All Rights Reserved.

Qualcomm and Snapdragon are trademarks or registered trademarks of Qualcomm Incorporated.
Other products and brand names may be trademarks or registered trademarks of their respective owners.

References in this presentation to “Qualcomm” may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable. Qualcomm Incorporated includes our licensing business, QTL, and the vast majority of our patent portfolio. Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of our engineering, research and development functions, and substantially all of our products and services businesses, including our QCT semiconductor business.

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patents are licensed by Qualcomm Incorporated.

Follow us on: [in](#) [X](#) [@](#) [v](#) [f](#)

For more information, visit us at qualcomm.com & qualcomm.com/blog

