YAML for Device Trees

Bruce Ashfield
Stefano Stabellini
Tomas Evensen

Linaro Connect
VIRTUAL • FALL 2021

Intro

Tomas Evensen

Linaro Connect
VIRTUAL • FALL 2021

Recap: Device Trees and System Device Trees

- **Device Trees (DTs) express HW information relevant to** Operating Environments
  - Used by ARM, MicroBlaze, PPC, RiscV, etc. to define HW that cannot be dynamically discovered
  - Used by Uboot, Linux, Xen, ATF and increasingly being used by RTOS vendors
- **Device Trees describes HW nodes and topologies**
  - **CPUs, memory and devices**
  - ***Traditional Device Trees are only describing the world seen from one Address Space***
- **Additional system level information is defined in System DT additions:**

1. DeviceTree.org *specification* and tooling additions
   - Describing multiple cpu clusters
   - Open source project Lopper to manipulate System Device Trees
2. AMP *configuration* information defined by OpenAMP project
   - Using the Device Tree `domains` section to specify AMP configuration
   - Specification of shared resources, such as pages for virtIO buffers
   - Aligning with hypervisor information (e.g. Xen Dom0-less configuration)

devicetree.org

XILINX arm Linaro Qualcomm Texas Instruments Mentor A Siemens Business NXP WIND KALRAY ST life.augmented NORDIC SEMICONDUCTOR

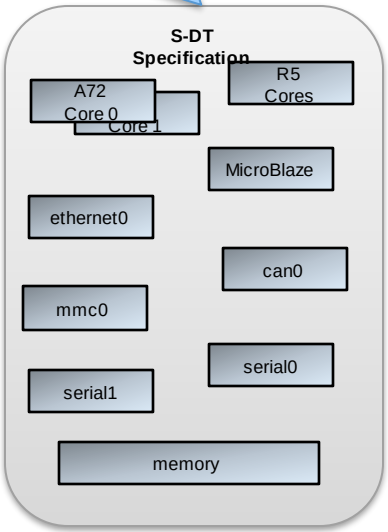
OpenAMP

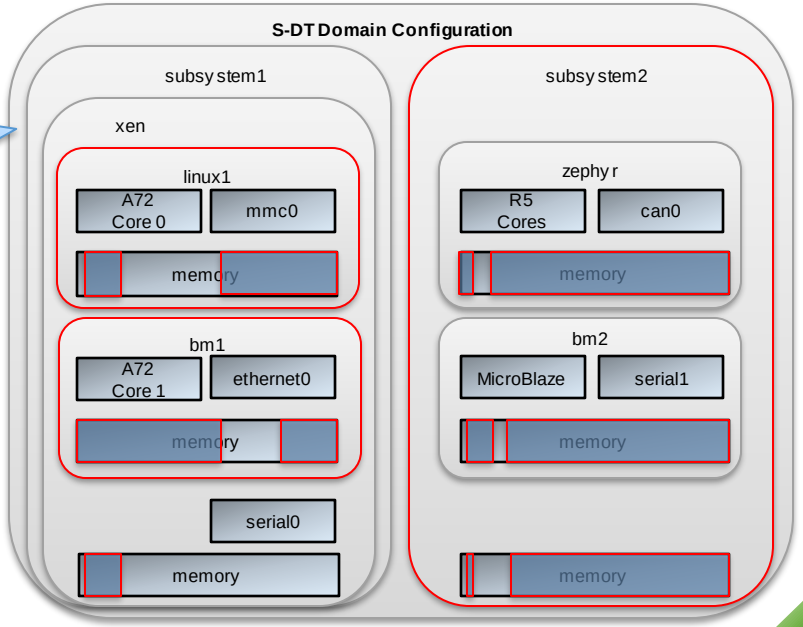
Linaro Connect VIRTUAL • FALL 2021

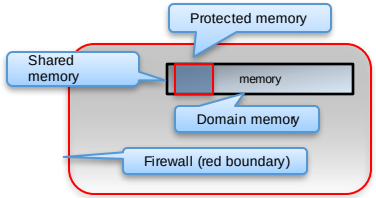
Allocating HW Resources to Domains

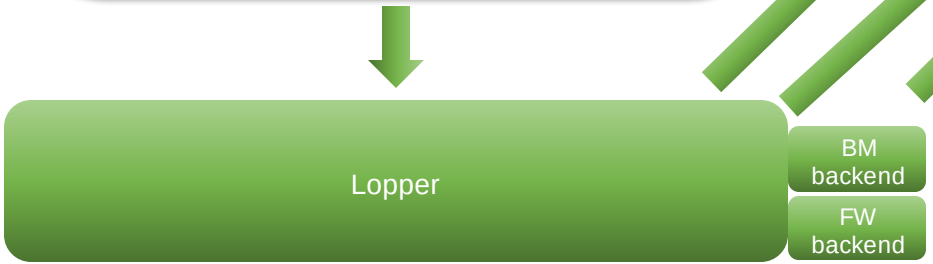
Domain configuration from SW Architect
- What HW goes where
- Domains: HW allocated to OS/FW/HV
- Subsystem: Domain(s) with common lifecycle
- Firewalls: Protection between domains
- Flag setting for domains

S-DT specification SOC/Board vendor
- HW description before any allocations
- CPUs, devices and memory
- Addresses, topologies, …









Runtime Clients

- Xen Device Tree
- Linux Device Tree (Traditional)
- Zephyr #defines
- Bare Metal #defines
- Firmware info subsystems/firewalls

Why YAML?

- More intuitive than Device Tree source
  - Easier to read and to write for humans and tooling
  - Especially important for domain information, as it is often defined by architects not necessarily familiar with DTs
- Functionally equivalent to DTS
- Opportunity for simplifications
  - Using attributes instead of mysterious hex bits
  - Automatically calculate information such as #size_cells
- Originally introduced for /domains, now extended to the entire DTS

YAML for Device Trees

Stefano Stabellini

Linaro Connect
VIRTUAL • FALL 2021

Nodes, Properties, and the Hierarchy

- Device Tree nodes are represented as YAML *mappings*
  - The content of the mapping corresponds to the node
  - The key of the mapping is the DTS node label (preferred) or node name
- When converting from YAML to DTS, node names are generated from YAML keys and compatible strings
- DTS properties are expressed in YAML as unordered *keys: value* pairs
- The Device Tree hierarchy is preserved in YAML by using appropriate indentation

Example

Device Tree source

```
amba {
    compatible = "simple-bus";

    can0: can@ff060000 {
        compatible = "xlnx,zynq-can-1.0";
        reg = <0x0 0xff060000 0x0 0x1000>;
    };
};
```

YAML

```
amba:
  compatible: simple-bus

  can0:
    compatible: xlnx,zynq-can-1.0
    reg:
      - start: 0xff060000
        size: 0x1000
```

Simplifications

- Simplified YAML goes beyond simply storing DTS information into YAML format
- It comes with a few simplifications to further enhance readability and intuitiveness of the source
- Simplifications:
  - strings
  - reg and address ranges
  - *_cells
  - phandles
  - booleans

Strings

Simplified YAML doesn't use quoted strings

Example:

```
compatible: openamp,domain-v1
```

reg and other address ranges

- List of address ranges (e.g. reg) are expressed as a YAML sequence of *key: value* pairs
  - start and size are keys
  - the start and size scalars are as large as needed; they are not broken down into 32-bit cells
  - addresses and sizes can also be expressed in human-readable formats, e.g. 2M, 4G, 1T.

Example:

```
reg:
  - start: 0xff060000
    size: 0x1000
  - start: 0x400000000
    size: 0x10000
  - start: 32G
    size: 1G
```

*_cells

- #address_cells, #size_cells, and other *_cells definitions are not used in Simplified YAML
- #address_cells and #size_cells are calculated as appropriate
- in general, the value of the related property is expressed as a *sequence* with a corresponding number of entries

Example:

```
interrupts:
  - [0x1, 0xd, 0xf08]
  - [0x1, 0xe, 0xf08]
  - [0x1, 0xb, 0xf08]
  - [0x1, 0xa, 0xf08]
```

Phandles

- Phandles are not used in Simplified YAML
- Only references are used
- The & is not used in Simplified YAML for references

Example:

```
interrupt-parent: interrupt-controller
```

Boolean Properties

- Boolean properties use true/false as values instead of 0x1/0x0
- Device Tree properties with no value in YAML become boolean *key: value* pairs with *true* value

Example:

```
status: okay
ranges: true
```

Example

```
compatible: xlnx,zynqmp-zcu102-rev1.0,
            xlnx,zynqmp-zcu102, xlnx,zynqmp
model: ZynqMP ZCU102 Rev1.0

cpus:
  cpu:
    compatible: arm,cortex-a53
    device_type: cpu
    enable-method: psci
    operating-points-v2: 0x1
    reg: 0x0
    cpu-idle-states: 0x2
    clocks: clock-controller 0xa

timer:
  compatible: arm,armv8-timer
  interrupt-parent: interrupt-controller
  interrupts:
    - [0x1, 0xd, 0xf08]
    - [0x1, 0xe, 0xf08]
    - [0x1, 0xb, 0xf08]
    - [0x1, 0xa, 0xf08]
```

```
amba:
  compatible: simple-bus
  ranges: true

  interrupt-controller:
    compatible: arm,gic-400
    reg:
      - start: 0xf9010000
        size: 0x10000
      - start: 0xf9020000
        size: 0x20000
      - start: 0xf9040000
        size: 0x20000
      - start: 0xf9060000
        size: 0x20000
    interrupt-controller: true
    interrupt-parent: interrupt-controller
    interrupts: [0x1, 0x9, 0xf04]
    num_cpus: 0x2
    num_interrupts: 0x60

  can0:
    compatible: xlnx,zynq-can-1.0
    status: okay
    clock-names: can_clk, pclk
    reg:
      - start: 0xff060000
        size: 0x1000
    interrupts: [0x0, 0x17, 0x4]
    interrupt-parent: interrupt-controller
    tx-fifo-depth: 0x40
    rx-fifo-depth: 0x40
```

TODO: Introducing new Simplifications

- Further simplifications will often require binding-specific information (e.g. reg)
- Each Device Tree binding should be able to specify YAML simplifications
  - a formal language for bindings (already written in YAML) to specify YAML simplifications

Example: interrupts property of ARM GICv2 interrupt controllers

FROM

```
interrupts:
  - [0x1, 0xd, 0xf08]
```

TO

```
interrupts:
   - type: PPI
     number: 0xd
     level: true
     trigger: low
     mask: 0xf
```

Lopper & YAML

Bruce Ashfield

Linaro Connect
VIRTUAL • FALL 2021

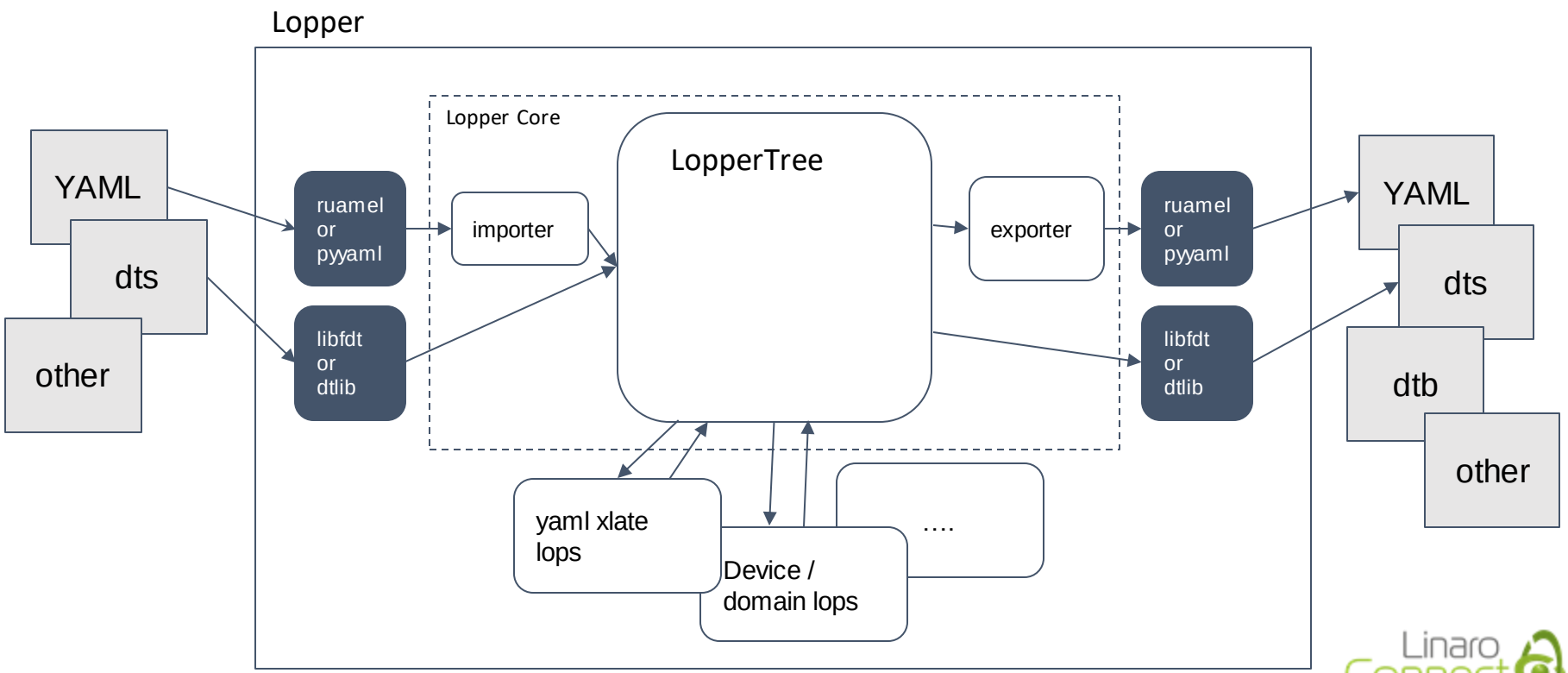
Overview: Lopper and Core Technology

- Lopper
  - Repository: **github.com/devicetree-org/lopper**
    - ~ 3 years of evolution and development
  - Tool for manipulating System Device Trees
  - Primary goal is to produce standard devices trees to support existing platforms/OSs
  - Produces any number of outputs: device trees, generated code, custom, etc
  - Integrates with development workflows
  - Data driven (there is no magic!)
- A few details:
  - OpenSource, BSD-3 License
  - Written in python, leveraging standard tools / libraries
    - dtc, libfdt, dtlib, etc
  - Works with dts, dtb and yaml inputs
  - Supports basic/simple operations (lops) and more complex python assist modules
    - Depending on the task, both can be used
  - Flexible output / input is provided via python assists
  - Performs validation and consistency checking

Lopper: A Framework

- Lopper as a Framework
  - Common source base for any tooling inquiring or manipulating device trees
  - Built-in:
    - core tree manipulation
    - dts/dtb/yaml input and output
    - libfdt or dtlib parsing
  - Optional / plugin:
    - Front ends
    - Backends
    - in depth tree analysis / modification
    - source validation and expansion
    - ReST API server

YAML expansion



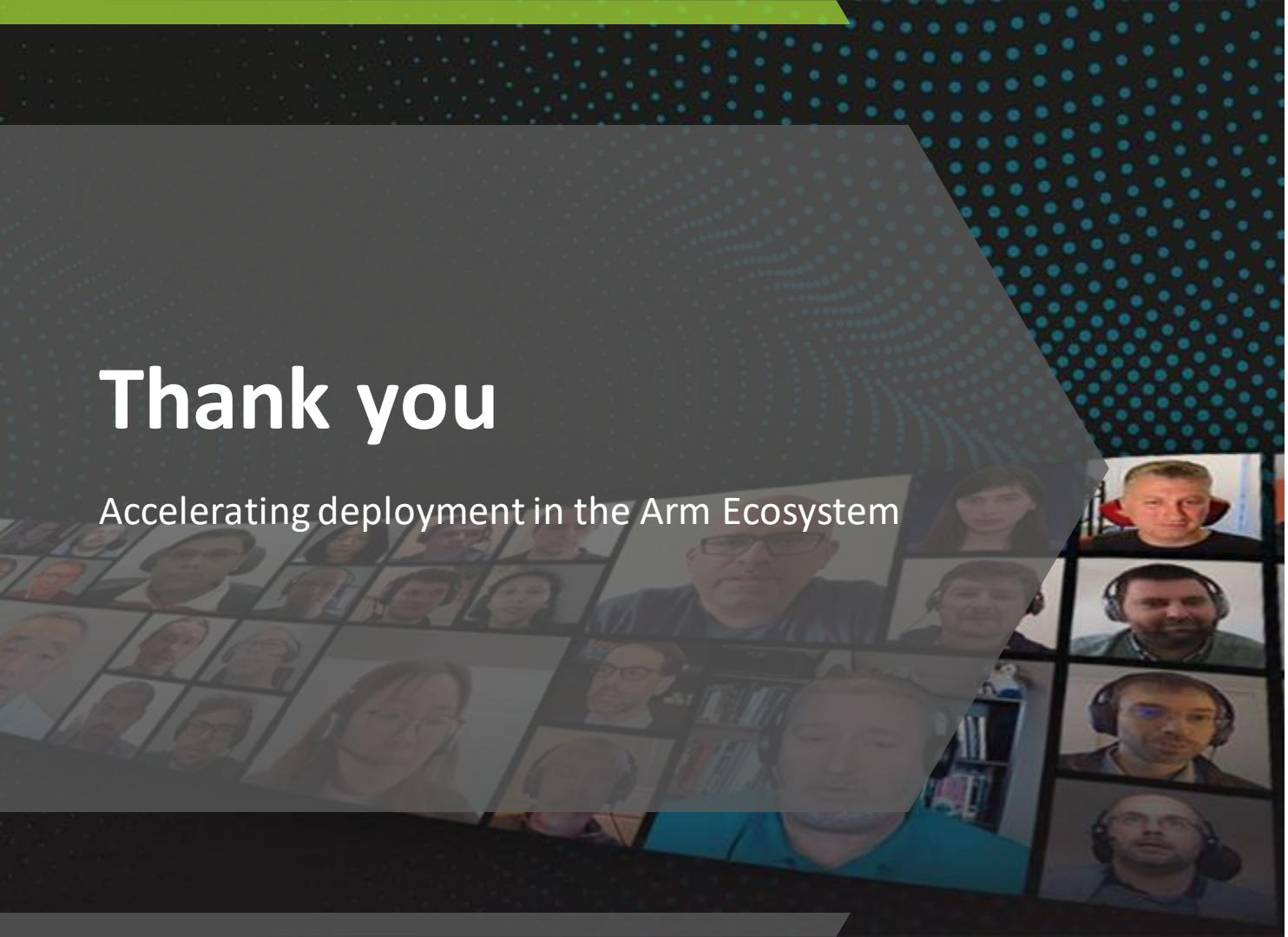
YAML: expansion examples

- Firewall:
  - Sample of the System Device Tree firewall specification
  - Processing Pipeline:
    - Standard import capabilities
    - yaml expansion lops
    - assist / custom domain lops

- Simplified device tree
  - Sample of the proposed, more concise device tree specification
  - Work in progress, is not complete
  - Processing pipeline:
    - Standard import
    - Standard yaml expansion
    - Placeholder for device tree logic / expansion

Thank you

Accelerating deployment in the Arm Ecosystem



Linaro Connect
VIRTUAL • FALL 2021