json-schema for Devicetree

Rob Herring

OCTO
Office of CTO

Devicetree Schema Documentation and Validation

The problem: too easy to get devicetree wrong

- Data must be encoded in very specific ways
- Toolchain provides little validation
- No checks against documented schema (aka. bindings)
  - Schemas are loosely structured prose
  - Not machine readable
- Too much manual review of bindings and dts
- Steep learning curve

Source: Grant Likely, Linaro Connect HKG18

Linaro OCTO Office of CTO

Prior Attempts

- C based schema in dtc (Stephen Warren, 2013),
  - https://www.spinics.net/lists/arm-kernel/msg282324.html
- Using dts as schema language (Benoit Cousson & Fabien Parent, 2013),
  - https://lwn.net/Articles/568217/
- Another using dts as schema language (Tomaz Figa, 2014)
- YAML, take 1 - custom syntax, no constraints (Matt Porter, Fall 2015)
- YAML, take 1.5 - custom syntax (Grant, LPC 2016)
- YAML, take 1.5 - eBPF (Pantelis, ELCE 2017)
- YAML, take 2 - json-schema (Grant, ELCE 2017)
  - Credit to Alison Chaiken?: http://lists.infradead.org/pipermail/linux-arm-kernel/2013-October/202507.html

    ‘The syntax of the existing device-tree source is strikingly similar to that of the widely used JavaScript Object Notation, better known as JSON. JSON has many parsers, validators and schemata already in existence. Assuredly as many coders know how to program JSON as know C.
    JSON makes some sense for representation of device-trees, because, as the authoritative json.org explains, "JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages . . . JSON is a natural representation of data for the C family of programming languages."’

Project Goals

- Define a DT schema language
  - Human friendly
  - Machine readable
  - Include binding documentation
- Better tooling
  - Validate DTS files at build time
  - Validate DT Schema files are in the correct format
  - Useful error and warning messages
- Leverage existing technology
  - Use existing schema validation framework
    - Extended to handle quirks of DT
  - Don’t write a lot of code!
  - Don’t define an entirely new language!
- Generate Specification Documentation from Schema files

Source: Grant Likely, Linaro Connect HKG18

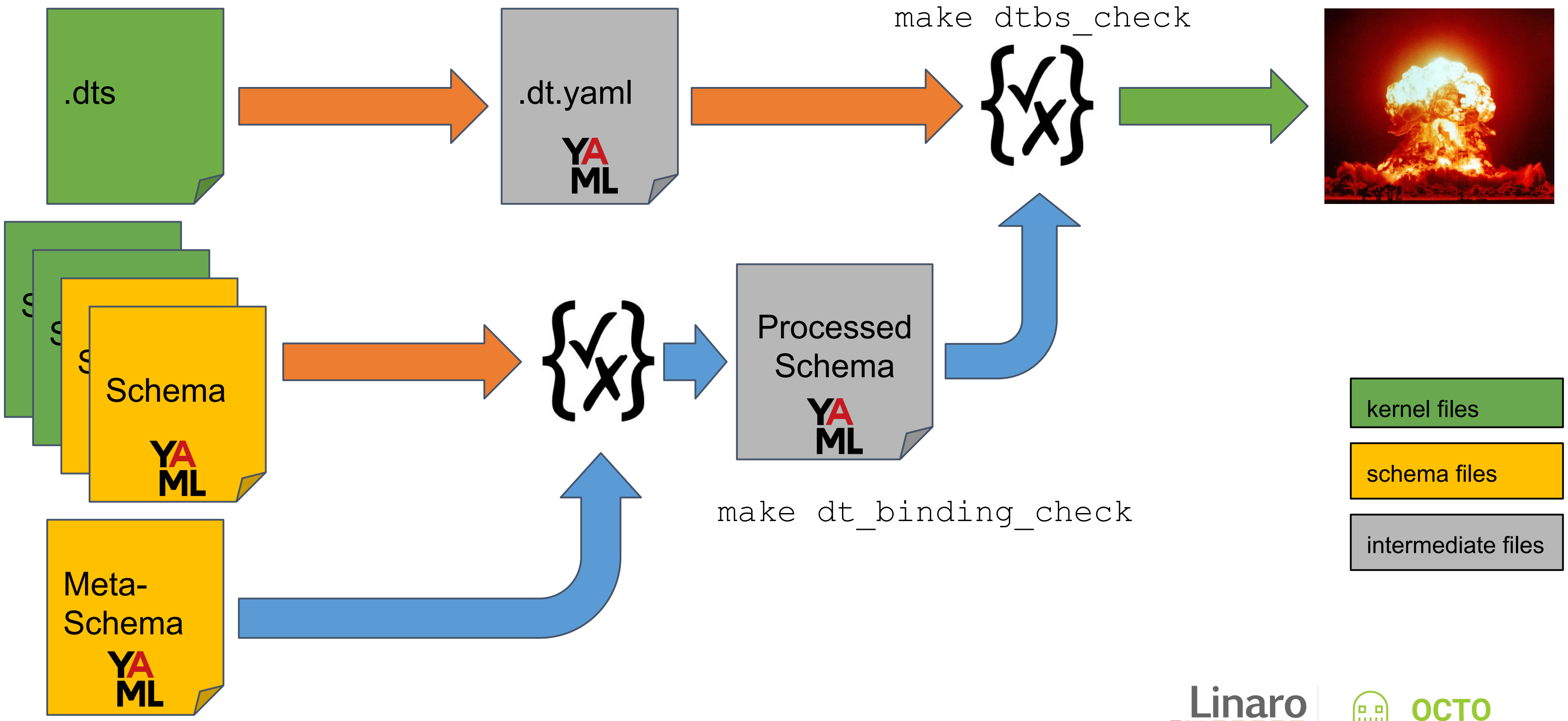
Current Status/Features

- Using json-schema Draft 6 for schema vocabulary
- Schema docs in YAML format
    - JSON compatible subset
    - Comments!
    - 1 schema per file (to keep 1-1 $id to filename mapping)
- dtc support for dts->YAML output
    - YAML tags for size and phandle typing
    - Validation format only, no stability guarantee
- Tools written in Python 3 utilizing ruamel.yaml and jsonschema modules
- Project repo: https://github.com/robherring/yaml-bindings

Current Status/Features, cont.

- schema/tools one step install from pip (Works for Me™)
- YAML output format support in upstream dtc
- Kernel build integration
  - schema doc validation
  - dts validation with schema - core schema and some ARM board level binding schema
  - Binding examples extracted from schema and built with dtc for validation
- Validation support for size (8-bit, 16-bit, etc.) and phandle tags
- Build time performance is reasonable now, but don't have 1000s of binding schemas yet

DT Schema Build Flow



🟩 kernel files

🟨 schema files

⬜ intermediate files

DT YAML Encoding

- Output supported in upstream dtc
- For validation only, format subject to change
- Strings always an array/list
- Scalars always a matrix
- dts '< >' are maintained
  - Will need to be stricter about these
  - Same for dtb, not for validation:
    - `prop = <1 2 3 4>;`
    - `prop = <1>, <2>, <3>, <4>;`
-

Example:

```
- model: [ "none" ]
  compatible: [ "none" ]
  "#address-cells": [[1]]
  "#size-cells": [[1]]

  good-test1:
    compatible: [ "vendor,soc4-ip", "vendor,soc1-ip" ]
    reg: [ [0x0, 0x4], [0x8, 0x4] ]
    reg-names: [ "coreAAA", "aux" ]
    interrupt-controller: true
    '#interrupt-cells': [[2]]
    interrupts: [ [10], [11, 1, 0] ]
    some-gpios: [ [ !phandle 0xdead, 0, 0 ] ]
    clocks: [ [0] ]
    vendor,bool-prop:
    vendor,int-prop: [[3]]
    vendor,string-prop: [ "foo" ]
    vendor,int8-prop: [ !u8 [1] ]
    vendor,int8-array-prop: [ !u8 [1, 2] ]
    vendor,int64-prop: [ !u64 [0x100000000] ]
    vendor,phandle-prop: [[!phandle 1]]
    vendor,phandle-array-prop: [[!phandle 1], [!phandle 2]]
```

Schema doc contents

- $id - URI with json-schema unique identifier (within a set of schemas)
- $schema - URI for meta-schema the schema adheres to
- title - A one-line description for the binding
- description - A multi-line description for the binding
- maintainers* - List of email addresses for owner(s) of the binding
- select* - Schema to match DT nodes. Only needed when not matching by compatible or node name.
- properties/patternProperties - dictionary of DT properties for a binding
- allOf - List of other schemas to include. Can reference common schema
- required - List of mandatory properties
- examples - List of examples in dts syntax

properties schema

- The primary part used in validation of DTs
- Contains list of DT property constraints for a binding
- Can also be child nodes with their own DT properties
- A subset of json-schema is allowed and checked by the meta-schema
- Common properties only need to define what a common schema cannot
  - common constraints: description, data type, range of values, range of elements
  - binding specific constraints: number of items, valid values, etc.
- Vendor specific properties need to reference base type
  - `$ref: /schemas/types.yaml#/definitions/<type>`

Common Property Examples

```
clock-frequency:
  minimum: 100
  maximum: 200
reg:
  items:
    - description: the first register range
    - description: the 2nd register range
  minItems: 2
  maxItems: 2
```

Vendor Property Examples

```
vendor,uint32-prop:
  allOf:
    - $ref: "/schemas/types.yaml#/definitions/uint32"
    - minimum: 100
      maximum: 200
  description: A vendor uint32 property
vendor,string-prop:
  allOf:
    - $ref: "/schemas/types.yaml#/definitions/string"
    - enum: [ foo, bar, baz ]
  description: A property with meaningless strings
```

Gotchas

- YAML is indentation sensitive and doesn’t like tabs
- json-schema keywords are case sensitive
- Validator handling of unknown json-schema keywords is to ignore
- Constraints dependent on other property’s data not easily expressed (e.g. 2 interrupts if compatible A or 1 interrupt if compatible B)
- Using allOf/oneOf/anyOf results in vague error messages
- Only one binding per doc. YAML could support more, but breaks json-schema $ref handling
-

Cross property dependencies

```
if:
  properties:
    compatible:
      contains: a-compatible-string
then:
  properties:
    a-conditional-prop:
      const: 30
else:
  properties:
    a-conditional-prop:
      const: 50
```

Note: if/then/else new in draft7 (only draft6 currently supported)

Running

- pip3 install git+https://github.com/robherring/yaml-bindings.git@master
- Install libyaml and its headers (for dtc)
- make allmodconfig
- make dt_binding_check
- make dtbs_check
    - Only core schemas
        - DT_SCHEMA_FILES=""
    - Only kernel schemas
        - DT_SCHEMA_FLAGS="-u"
    - Only user specified schema(s)
        - DT_SCHEMA_FLAGS="-u"
        - DT_SCHEMA_FILES="Documentation/devicetree/bindings/${schema}.yaml"

Next Steps

- Review json-schema patch series: https://lkml.org/lkml/2018/10/5/883
- Adding schema validation build support in v4.21
    - Build support is ~70 lines and separate target from normal builds
    - Some bindings converted. Mostly ARM top level board/SoC bindings
    - No requirement to submit bindings in YAML (yet)
- 1000s of warnings to fix already - Help wanted!
    - Lots of duplication due to warnings repeating for every board including an SoC .dtsi
    - Need a 'build SoC only DTs' option
- 3000 schema is going to be slow...
- Convert bindings - Help wanted!

Open Questions

- What to do with yaml-bindings project?
    - Keep separate
    - Integrate into dtc
    - Integrate a subset into kernel
- Changes to make targets
    - Separate targets or sparse like “make C=1 dtbs”?
- How to define class of devices (e.g. an “I2C device”, “touchscreen device”, etc.)
    - Just include another schema doc in a schema doc:
        - `allOf: [ {$ref: /schemas/class.yaml#}, {$ref: /schemas/bus-device.yaml#} ]`
        - Can be extended if device supports multiple classes (e.g. clock and reset controller)
        - Or any other type of schema we haven’t thought of yet
    - Needed for common bindings? GPIO, clocks, etc.
- Licensing
    - Binding docs are default GPLv2
    - yaml-bindings proj is BSD 2 clause (still easy to change)

Resources

- Schema/Tools repo
  - https://github.com/robherring/yaml-bindings
- kernel repo with DT schema
  - https://github.com/robherring/linux/tree/yaml-bindings
-