Zephyr RTOS & System devicetree

Martí Bolívar | 2023-02-21

About me

- Nordic Semiconductor employee
- Zephyr devicetree co-maintainer
- Working on system devicetree support in Zephyr
- Contributing to lopper/specification

About this talk

- What is Zephyr?
- How Zephyr uses devicetree
- Why it's not enough anymore
- Why we think system devicetree can help
- What we've done so far
- What's left to do

Zephyr (so far) in 60 seconds

https://lwn.net/Articles/824029/

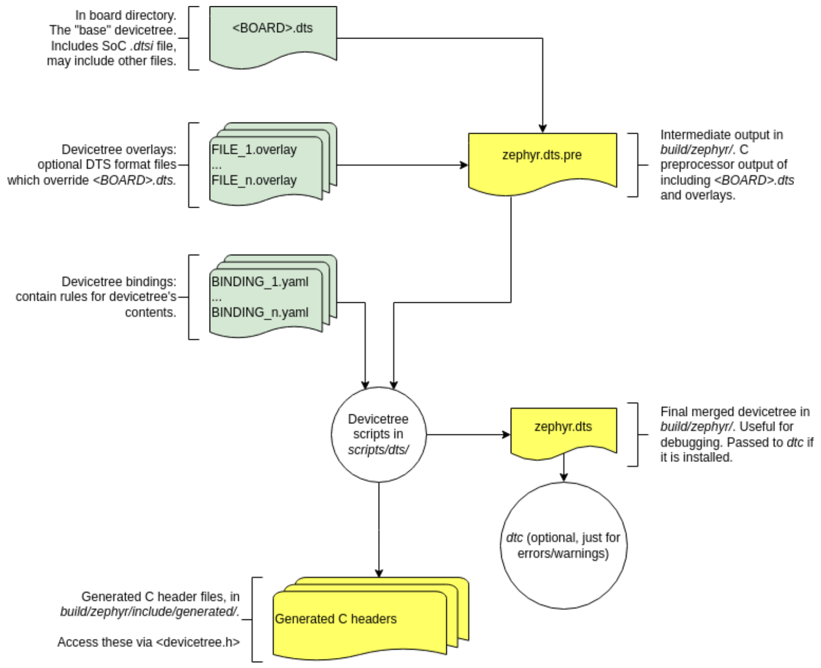
- Typical target has been an MCU with:
  - <=100 MHz CPU
  - <=512 KB flash
  - 32-256 KB SRAM
- Reuses Linux technologies
  - Kconfig
  - Devicetree

What Zephyr uses devicetree for

- Allocating struct devices: 100% at **build** time
- Configuring individual device boot time behavior
- Influencing which Kconfig options are available and their defaults
- Setting up memory regions
- Miscellaneous structured configuration

Devicetree dataflow in Zephyr

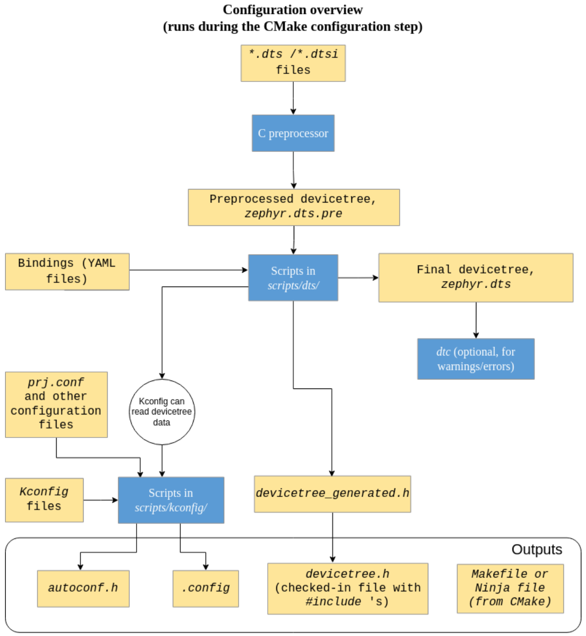
https://docs.zephyrproject.org/latest/build/dts/index.html


In board directory. The "base" devicetree. Includes SoC .dtsi file, may include other files.
<BOARD>.dts
Devicetree overlays: optional DTS format files which override <BOARD>.dts.
FILE_1.overlay
...
FILE_n.overlay
zephyr.dts.pre
Intermediate output in build/zephyr/. C preprocessor output of including <BOARD>.dts and overlays.
Devicetree bindings: contain rules for devicetree's contents.
BINDING_1.yaml
...
BINDING_n.yaml
Devicetree scripts in scripts/dts/
zephyr.dts
Final merged devicetree in build/zephyr/. Useful for debugging. Passed to dtc if it is installed.
dtc (optional, just for errors/warnings)
Generated C header files, in build/zephyr/include/generated/.
Access these via <devicetree.h>
Generated C headers

- No dtc: pure Python devicetree implementation
- No fdt: DTS → #defines
- No runtime access: devicetree only available at build time

Devicetree in the larger build system

https://docs.zephyrproject.org/latest/build/cmake/index.html


Configuration overview
(runs during the CMake configuration step)
.dts /.dtsi files
C preprocessor
Preprocessed devicetree, zephyr.dts.pre
Bindings (YAML files)
Scripts in scripts/dts/
Final devicetree, zephyr.dts
dtc (optional, for warnings/errors)
prj.conf and other configuration files
Kconfig can read devicetree data
Kconfig files
Scripts in scripts/kconfig/
devicetree_generated.h
Outputs
autoconf.h
.config
devicetree.h (checked-in file with #include 's)
Makefile or Ninja file (from CMake)

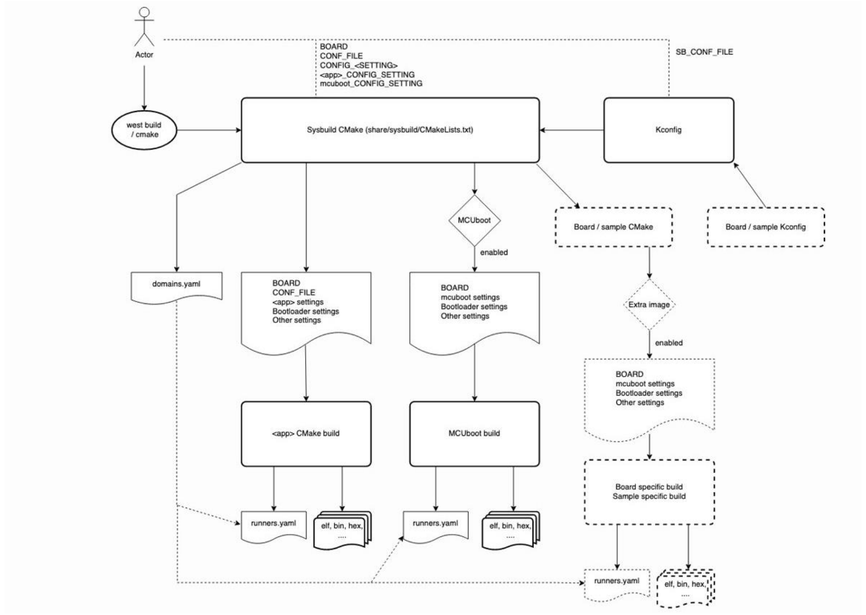
- "Full custom" DT tooling influences "just about everything"
  - Tightly coupled to complex build system internals
  - We have a devicetree API **in cmake** too that constrains the generated build system
- Custom bindings language
  - Hopefully DTSchema instead eventually
- **Faithful implementation of DTSpec**

Problem statement

- This worked for a while, but stopped scaling
- Multi-core AMP SoCs not well supported
- AMP multi-core Arm v8-M MCUs with TrustZone support: the last straw
  - Memory addressing in v8-M: peripheral addresses vary by security state
  - Duplicated static memory allocations in different build systems
  - Shared IPC resources for coprocessors is a pain too
  - …
- It's only going to get worse

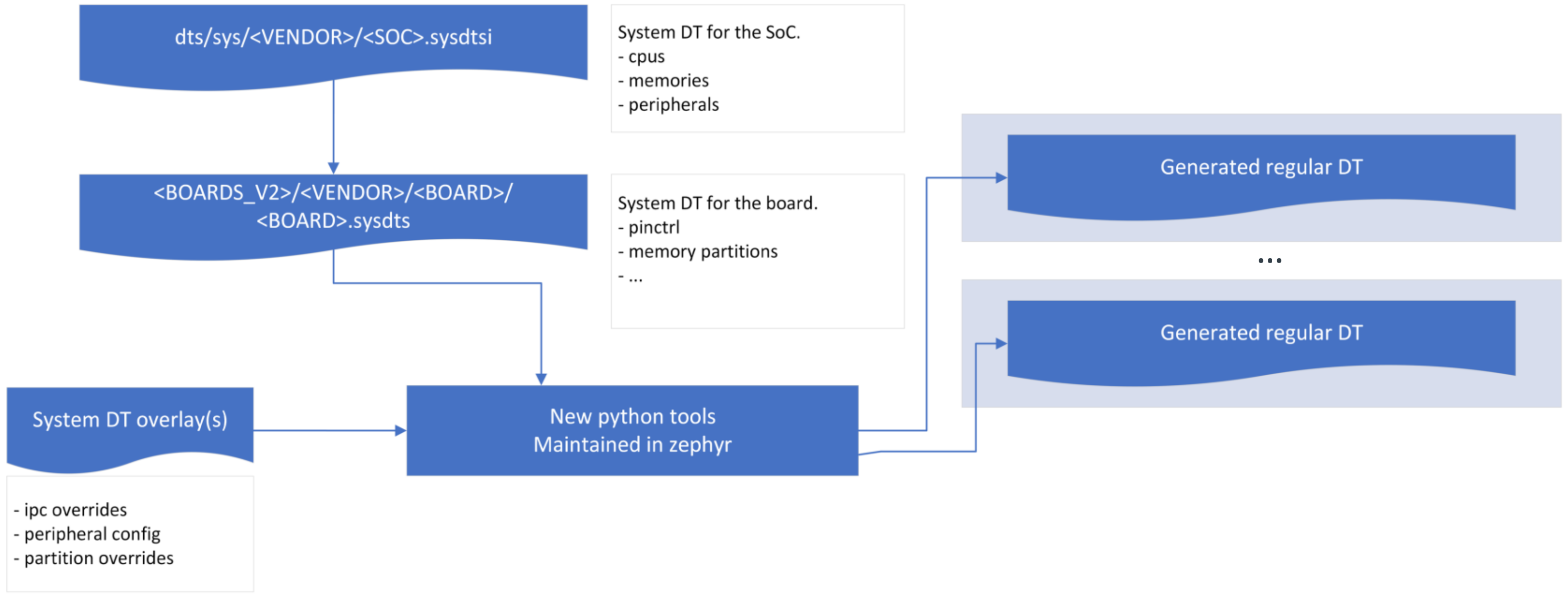
Scaling the build system: sysbuild

https://docs.zephyrproject.org/latest/build/sysbuild/index.html



- This is our meta-build thing
- "Parent" CMake build system (sysbuild) spawns, configures, runs individual Zephyr build systems
- Every build system has its own DTS, .config, etc.
- … maybe you can see where this is going

Sysbuild and system devicetree



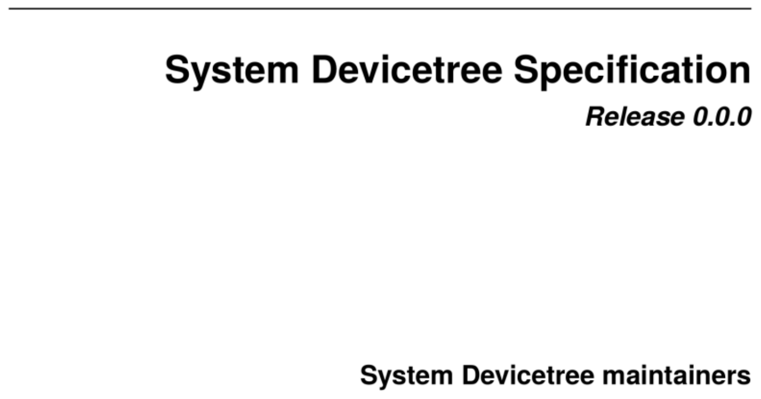
Our approach

https://github.com/zephyrproject-rtos/zephyr/issues/51830

- Work closely with upstream to flesh out the System DT specification
  - Not interested in forking the spec
  - Want to make sure our implementation matches the spec
- Faithfully implement the system DT spec in our custom tools
  - We already have Python-based DT manipulation, our own internal conventions for managing the preprocessor, our own hairy build systems, etc.
  - Extra power and flexibility lopper provides is not currently needed for our use cases; simpler to extend what we have
  - Leaving option open to adopt lopper as well in future if our needs outgrow our tools

Contributions

- System DT specification converted to Sphinx format
- Format used by DTSpec and the Linux kernel docs
- HTML and PDF builds



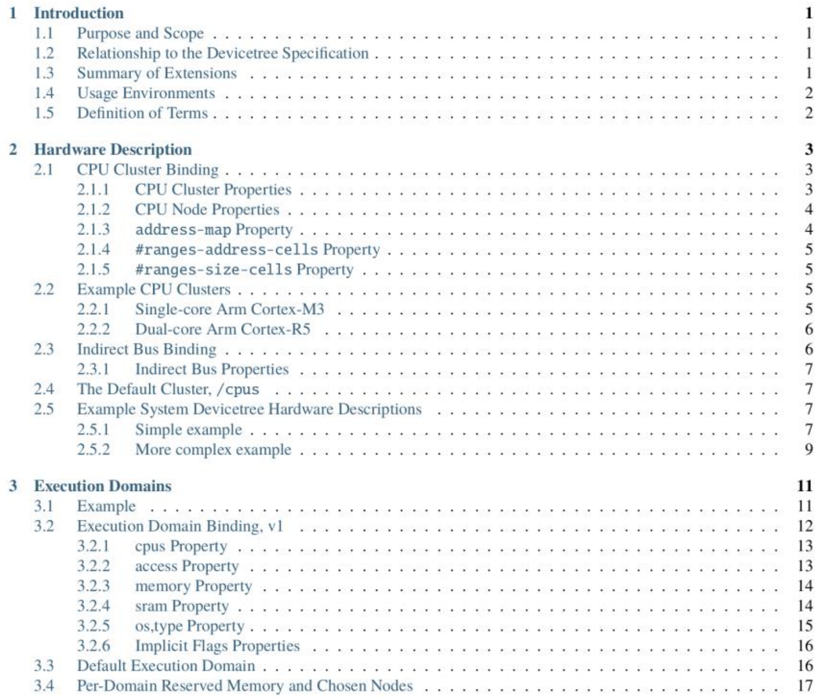
System Devicetree Specification

Release 0.0.0

System Devicetree maintainers

Contributions

- Style and content of specification reworked to match DTSpec
- More examples, tables of properties, etc.
- Thanks to Stefano and Bruce for all the reviews and clarifications!



The road ahead

- Collaborate to finalize system DT specification v1.0
  - Bruce has agreed to start tagging spec releases, starting with v0.9
  - I will be opening issues for remaining spec questions I have and continuing to post patches to close issues as I discover them while implementing, with help from Stefano and Bruce
- Many, many internal Zephyr community reviews and discussions
- …
- Profit! System DT adopted in Zephyr and used as the DT layer within sysbuild
- (Probably some feedback loop towards system DT v2.0 after that)