Zephyr™ OS Configuration via Device Tree

Andy Gross - Linaro IoT

Configuration in Zephyr today

- Configuration is spread out across the system.
- Most configuration is hardcoded.
- Difficult to deal with device multiples.
- Definitions come from multiple file sources, (CMSIS, vendor includes, etc)
- Not extensible for similar boards or SoCs.

Using device tree for Zephyr OS configuration

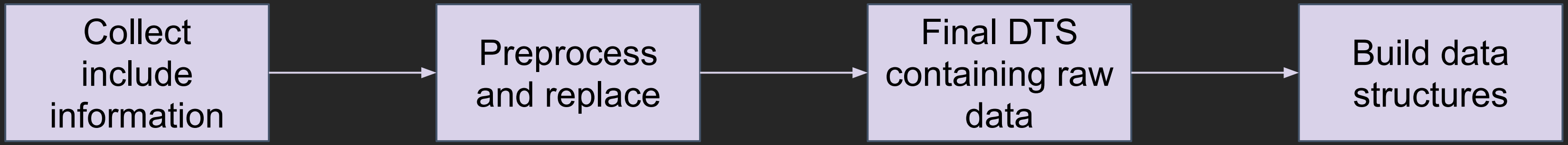
- Device tree is architecturally neutral
- Less need for Kconfig options as specific config comes from DTS
- Device tree can describe any device node
- Device description is extensible
- Other layers could use device tree information (apps, hal, etc)
- Adding new boards/SoCs is easier

Required tooling for device tree usage

- Use the available configuration sources where applicable (CMSIS, vendor files, etc)

- Use the C preprocessor to leverage those configuration sources

- Build the target configuration from the processed device tree information



Collect include information

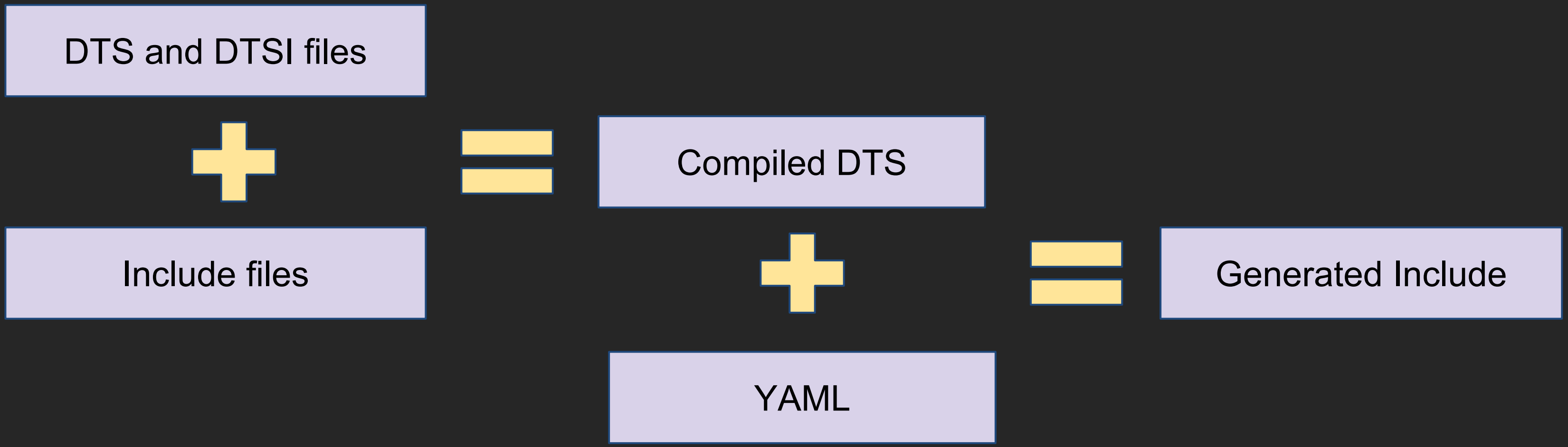
Preprocess and replace

Final DTS containing raw data

Build data structures

*Other names and brands may be claimed as the property of others.

Generating include files



Using YAML in Zephyr

- Devices are described in DT and YAML.
- YAML gives a description of the contents of the node
  - Definitions for properties
  - Targets for extraction
  - Format for output
- Allows for validation of DT contents.

YAML / DT Example

```
---
inherits:
  - !include uart.yaml
  - !include zephyr_devices.yaml

properties:
  - compatible:
      type: string
      category: required
      description: compatible strings
      constraint: "arm,cmsdk-uart"

  - reg:
      type: array
      description: mmio register space
      generation: define
      category: required

  - interrupts:
      type: array
      category: required
      description: required interrupts
      generation: define
...
```

```
uart0: uart@40004000 {
        compatible = "arm,cmsdk-uart";
        reg = <0x40004000 0x14>;
        interrupts = <0>;
        zephyr,irq-prio = <3>;
        baud-rate = <115200>;
 };
```

Generated Output

```
/* uart@40004000 */
#define ARM_CMSDK_UART_40004000_BASE_ADDRESS_0          0x40004000
#define ARM_CMSDK_UART_40004000_BAUD_RATE               115200
#define ARM_CMSDK_UART_40004000_IRQ_0                   0
#define ARM_CMSDK_UART_40004000_SIZE_0                  20
#define ARM_CMSDK_UART_40004000_ZEPHYR_IRQ_PRIO         3
#define ARM_CMSDK_UART_40004000_BASE_ADDRESS            ARM_CMSDK_UART_40004000_BASE_ADDRESS_0
#define ARM_CMSDK_UART_40004000_SIZE                    ARM_CMSDK_UART_40004000_SIZE_0

/* Fixup */

#define CMSDK_APB_UART_0_IRQ                            ARM_CMSDK_UART_40004000_IRQ_0
#define CONFIG_UART_CMSDK_APB_PORT0_IRQ_PRI             ARM_CMSDK_UART_40004000_ZEPHYR_IRQ_PRIO
#define CONFIG_UART_CMSDK_APB_PORT0_BAUD_RATE           ARM_CMSDK_UART_40004000_BAUD_RATE
```

Current state of development

- Device tree support now in Zephyr 1.7.0.
- DTS python parsing script/library is now part of Zephyr
- Additional Python scripts generate the include information from the DTS->DTS passthrough
- YAML used to describe contents of device nodes
- Using temporary fixup file to map generated data to driver instances
- Support for ARM Beetle, TI CC3200, STM32L476RG, and NXP Kinetis

Work for the near term

- Cleanup the configuration directories for the boards as the required existing config and board files are retired. This will most likely involve complete removal of the board/ directories.

- Leverage the generated files and use this information to initialize drivers. This is ongoing.

- Generate overarching config options for devices based on DT status.

- Add platform data and structure support.

Questions?