Zephyr usage of Device Tree

Problem in Zephyr

- Zephyr is an RTOS aimed at the microcontroller space
  - https://www.zephyrproject.org
  - https://github.com/zephyrproject-rtos/zephyr
  - Supports multiple architectures (ARM, x86, ARC, NIOS II, RISC-V, Xtensa)

- Kconfig used to describe hardware/device properties (IRQ, IRQ priority, base address, etc)
  - Bit of a mess
  - Configuration the user should never see
  - Didn’t scale - dealing with systems with lots of peripherals (lots of duplication)
  - Obvious solution … Device tree!!!

Obvious Solution… Device Tree!!!

- Introduce Device Tree usage for microcontrollers
  - Not actually new, Linux Kernel has device tree for several mmu-less SoCs (STM32, LPC, etc)
- DTB is too big, but we can generate code based on .dts
- Initial work by Andy Gross (Linaro) and Leandro Pereira (Intel). Utilizing work from Grant Likely

What do we do

- Generate code utilizing dts and binding described in YAML
  - dts passed through CPP and dtc to sanitize
- Code generation written as a python script
- We parse through the dts and match compatibles to YAML bindings
- YAML utilized by generator to provide structure to the data

Describe bindings in YAML (ns16550.yaml - part 1)

```
---
title: ns16550
id: ns16550
version: 0.1

description: >
    This binding gives a base representation of the ns16550 UART

inherits:
  - !include uart.yaml

properties:
  - compatible:
      type: string
      category: required
      description: compatible strings
      constraint: "ns16550"
```

Describe bindings in YAML (ns16550.yaml - part 2)

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

  - reg-shift:
      type: int
      category: optional
      description: quantity to shift the register offsets by
      generation: define

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

Describe bindings in YAML (uart.yaml)

```
---
title: Uart Base Structure
id: uart
version: 0.1

description: >
    This binding gives the base structures for all UART devices

properties:
  - clock-frequency:
      type: int
      category: optional
      description: Clock frequency information for UART operation
      generation: define
  - current-speed:
      type: int
      category: required
      description: Initial baud rate setting for UART
      generation: define
```

What we generate - Kconfig

```
CONFIG_SRAM_BASE_ADDRESS_0=0x20000000
CONFIG_SRAM_SIZE_0=192
CONFIG_SRAM_BASE_ADDRESS=0x20000000
CONFIG_SRAM_SIZE=192
CONFIG_FLASH_BASE_ADDRESS_0=0x0
CONFIG_FLASH_LOAD_OFFSET=0
CONFIG_FLASH_LOAD_SIZE=0
CONFIG_FLASH_SIZE_0=1024
CONFIG_FLASH_BASE_ADDRESS=0x0
CONFIG_FLASH_SIZE=1024
CONFIG_UART_CONSOLE_ON_DEV_NAME="UART_0"
CONFIG_UART_PIPE_ON_DEV_NAME="UART_0"
```

What we generate - header defines

```
adc0: adc@4003b000{
        compatible = "nxp,kinetis-adc16";
        reg = <0x4003b000 0x70>;
        interrupts = <39 0>;
        label = "ADC_0";
        status = "disabled";
};
```

Becomes：

```
/* adc@4003b000 */
#define NXP_KINETIS_ADC16_4003B000_BASE_ADDRESS_0       0x4003b000
#define NXP_KINETIS_ADC16_4003B000_IRQ_0                39
#define NXP_KINETIS_ADC16_4003B000_IRQ_0_PRIORITY       0
#define NXP_KINETIS_ADC16_4003B000_LABEL                "ADC_0"
#define NXP_KINETIS_ADC16_4003B000_SIZE_0               112
#define NXP_KINETIS_ADC16_4003B000_BASE_ADDRESS         NXP_KINETIS_ADC16_4003B000_BASE_ADDRESS_0
#define NXP_KINETIS_ADC16_4003B000_SIZE                 NXP_KINETIS_ADC16_4003B000_SIZE_0
```

What we generate - header defines

```
i2c0: i2c@40066000 {
        compatible = "nxp,kinetis-i2c";
        clock-frequency = <I2C_BITRATE_STANDARD>;
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x40066000 0x1000>;
        interrupts = <24 0>;
        clocks = <&sim KINETIS_SIM_BUS_CLK 0x1034 6>;
        label = "I2C_0";
        status = "disabled";
};
```

Becomes:

```
/* i2c@40066000 */
#define NXP_KINETIS_I2C_40066000_BASE_ADDRESS_0         0x40066000
#define NXP_KINETIS_I2C_40066000_BASE_ADDRESS_1         0x1000
#define NXP_KINETIS_I2C_40066000_CLOCK_FREQUENCY        100000
#define NXP_KINETIS_I2C_40066000_IRQ_0                  24
#define NXP_KINETIS_I2C_40066000_IRQ_0_PRIORITY         0
#define NXP_KINETIS_I2C_40066000_LABEL                  "I2C_0"
#define NXP_KINETIS_I2C_40066000_BASE_ADDRESS           NXP_KINETIS_I2C_40066000_BASE_ADDRESS_0
```

Fixup files:

- Fixup files used to “normalize” names

```
#define CONFIG_I2C_0_NAME                    NXP_KINETIS_I2C_40066000_LABEL
#define CONFIG_FXOS8700_I2C_NAME             NXP_KINETIS_I2C_40066000_LABEL
#define CONFIG_I2C_MCUX_0_BASE_ADDRESS       NXP_KINETIS_I2C_40066000_BASE_ADDRESS_0
#define CONFIG_I2C_MCUX_0_IRQ                NXP_KINETIS_I2C_40066000_IRQ_0
#define CONFIG_I2C_MCUX_0_IRQ_PRI            NXP_KINETIS_I2C_40066000_IRQ_0_PRIORITY
#define CONFIG_I2C_MCUX_0_BITRATE            NXP_KINETIS_I2C_40066000_CLOCK_FREQUENCY
```

What’s next:

- Support buses (i2c, spi, etc)
  - Need to know parent/child relationship
- Support counting:
  - How many dma channels do I have
  - What’s the max IRQ on a given interrupt-controller
- Generate device structures instead of defines
- Support more device classes
- Convert other arch’s in Zephyr (used on ARM & x86 today)
- Unify device tree binding YAMLs with larger community effort