Using Device Tree with Zephyr

Erwan Gouriou
Andy Gross
Kumar Gala
Agenda

- DTS Usage in Zephyr / Code Generation [Erwan]
- TFM DTS Needs [Andy/Kumar]
- DT Miscellaneous Issues [Kumar]

Thur @ 10a
- YVR18-404:BoF Device Tree: Json-schema for DeviceTree bindings
Agenda

- Why introducing Device Tree in Zephyr?
- Kconfig, Device tree pro’s and con’s
- How Device Tree is used in Zephyr today
- Device Tree based #define’s generation
  - Process
  - Issues and Limitations
- Zephyr driver instantiation code
- Codegen introduction and first learnings
- Device Tree based code instantiation generation
- codegen related open points
- DT based code generation open points
Zephyr drivers instantiation code

```c
#ifdef CONFIG_I2C_4
static const struct i2c_config i2c_dev_cfg_4 = {
  .i2c = (I2C_TypeDef*)CONFIG_I2C_4_BASE_ADDRESS,
#ifdef CONFIG_I2C_STM32_INTERRUPT
  .irq_config_func = irq_config_func_4,
#endif
  .bitrate = CONFIG_I2C_4_BITRATE,
};
static struct i2c_data i2c_dev_data_4;

And so on for other instances ...

#ifdef CONFIG_I2C_STM32_INTERRUPT
static void irq_config_func_4(struct device *dev)
{
  IRQ_CONNECT(CONFIG_I2C_4_EVENT_IRQ,
  CONFIG_I2C_4_EVENT_IRQ_PRI,
  stm32_i2c_event_isr, DEVICE_GET(i2c_stm32_4), 0);
  irq_enable(CONFIG_I2C_4_EVENT_IRQ);
}
#endif

DEVICE_AND_API_INIT(i2c_stm32_4,
  CONFIG_I2C_4_NAME, &i2c_stm32_init,
  &i2c_dev_data_4, &i2c_dev_cfg_4, POST_KERNEL,
  CONFIG_KERNEL_INIT_PRIORITY_DEVICE,
  &api_funcs);
#endif /* CONFIG_I2C_4 */
```
Zephyr drivers instantiation code

```c
#ifdef CONFIG_I2C_4
static const struct i2c_config i2c_dev_cfg_4 = {
    .i2c = (I2C_TypeDef *)CONFIG_I2C_4_BASE_ADDRESS,
#ifdef CONFIG_I2C_STM32_INTERRUPT
    .irq_config_func = irq_config_func_4,
#endif
    .bitrate = CONFIG_I2C_4_BITRATE,
};
static struct i2c_data i2c_dev_data_4;

static void irq_config_func_4(struct device *dev)
{
    IRQ_CONNECT(CONFIG_I2C_4_EVENT_IRQ, CONFIG_I2C_4_EVENT_IRQ_PRI,
                stm32_i2c_event_isr, DEVICE_GET(i2c_stm32_4), 0);
    irq_enable(CONFIG_I2C_4_EVENT_IRQ);
}
#endif /* CONFIG_I2C_4 */

DEVICE_AND_API_INIT(i2c_stm32_4,
CONFIG_I2C_4_NAME,
&i2c_stm32_init,
&i2c_dev_data_4, &i2c_dev_cfg_4, POST_KERNEL,
CONFIG_KERNEL_INIT_PRIORITY_DEVICE,
&api_funcs);
#endif CONFIG_I2C_4
```
Why introducing Device Tree in Zephyr?

- Zephyr initial configuration system was based on Kconfig
  - Difficult to establish rules
  - Wild...
- Aim of DT introduction was to reduce configuration complexity due to extensive use of KConfig
- Initial introduction by Andy Gross:
  - Presentation at ELC-US on 17'02
  - https://www.youtube.com/watch?v=eOZ0_pNU5v
Kconfig pro’s and con’s

● Quite versatile, could be used in many ways:
  ○ Configuration: base_address = CONFIG_.....
  ○ Code Compilation: #ifdef CONFIG_
  ○ Build control: zephyr_library_sources_ifdef(CONFIG_USERSPACE uart_handlers.c)
  ○ Even used in linker.ld files...

● Comes with a UI that eases project configuration update by user

● No specification
  ○ Configuration spread across multiples files and folders (arch, boards, drivers)
  ○ Hard to control duplicated information
  ○ Difficult to instantiate common vocabulary
  ○ Difficult reuse across similar SoCs
Device Tree pro’s and con’s

● Made for HW description and boot time configuration
● Comes with a specification
  ○ HW description is modelized with a known vocabulary and grammar. It is then predictable and decoding could be automated
  ○ Well documented
  ○ Information gathered in a limited set of known places: .dtsi, .dts, .overlays, bindings
  ○ Information factorization is easy with inheritance model

● No UI for configuration or amending existing configuration
How device tree is used in Zephyr today

- Used in precompilation
  - Helps to minimize SW footprint
- Generates information for driver instances configuration
  - \#define’s
  - Device Instance base address, IRQ number, IRQ prio
  - Any useful information for boot time device instance configuration
- Relies on zephyr specific yaml binding files
  - Yaml files bound to a device thanks to compatible
  - Bindings help to specify:
    - what dts pieces should be interpreted
    - how information should be generated
- Fully functional on wide range of devices
  - Simple devices: Serial, SPI, I2C
  - Buses topology (sensors on busses, ...)
- More complex IP requires additional work:
  - clocks, pinmux, external interrupt
Device Tree based #define generation

- `.dtsi`/`.dts`
- `.overlay`
- `dt-bindings`
- `dtc`
- `.dt_compiled`
Device Tree based #define generation

- .dti/.dts
- .overlay
- dt-bindings
- dtc
- .dt_compiled
- devictree.py
- python dict
Device Tree based \#define generation

```
title: STM32 UART
type: uart

[...]
inherits:
  !include uart.yaml
properties:
  compatible:
    constraint: "st,stm32-uart"
reg:
  type: array
description: mmio register space
generation: define
category: required
interrupts:
  type: array
category: required
description: required interrupts
generation: define
```
Device Tree based #define generation

- .dtsi/.dts
- .overlay
- dt-bindings
- dtc
- .dtCompiled
- extract_dts_include.py
- .yaml
- devictree.py
- python dict
Device Tree based \#define generation

```
#define ST_STM32_USART_40004400_BASE_ADDRESS 0x40004400
#define ST_STM32_USART_40004400_CURRENT_SPEED 115200
#define ST_STM32_USART_40004400_IRQ_0 38
#define ST_STM32_USART_40004400_IRQ_0_PRIORITY 0
#define ST_STM32_USART_40004400_LABEL "UART_2"
```
# define CONFIG_UART_STM32_USART_2_BASE_ADDRESS ST_STM32_USART_40004400_BASE_ADDRESS
# define CONFIG_UART_STM32_USART_2_BAUD_RATE ST_STM32_USART_40004400_CURRENT_SPEED
# define CONFIG_UART_STM32_USART_2_IRQ_PRI ST_STM32_USART_40004400_IRQ_0_PRIORITY
# define CONFIG_UART_STM32_USART_2_NAME ST_STM32_USART_40004400_LABEL
# define USART_2_IRQ ST_STM32_USART_40004400_IRQ_0
Device Tree based #define generation

```c
#define CONFIG_UART_STM32_USART_2_BASE_ADDRESS ST_STM32_USART_40004400_BASE_ADDRESS
#define CONFIG_UART_STM32_USART_2_BAUD_RATE ST_STM32_USART_40004400_CURRENT_SPEED
#define CONFIG_UART_STM32_USART_2_IRQ_PRI ST_STM32_USART_40004400_IRQ_0_PRIORITY
#define CONFIG_UART_STM32_USART_2_NAME ST_STM32_USART_40004400_LABEL
#define USART_2_IRQ ST_STM32_USART_40004400_IRQ_0
#define ST_STM32_USART_40004400_BASE_ADDRESS 0x40004400
#define ST_STM32_USART_40004400_CURRENT_SPEED 115200
#define ST_STM32_USART_40004400_IRQ_0 38
#define ST_STM32_USART_40004400_IRQ_0_PRIORITY 0
#define ST_STM32_USART_40004400_LABEL "UART_2"
```
Device Tree based #define generation cont’d

- Flash partition
  - http://docs.zephyrproject.org/devices/dts/flash_partitions.html
  - ‘chosen’: Bind a hardware instance with software component:
    - zephyr,console = &usart1;
    - #define CONFIG_UART_CONSOLE_ON_DEV_NAME    "USART_1"

- Bus: yaml binding heuristics
  - Establish a master/slave relationship on a bus
  - #define ST_STM32_SPI_FIFO_40003C00_ST_SPBTLE_RF_0_BUS_NAME    "SPI_3"

- ‘alias’: Associate a generic #define to a particular node #define
  - bt = &bt0;
  - #define BT_BUS_NAME    ST_STM32_SPI_FIFO_40003C00_ST_SPBTLE_RF_0_BUS_NAME

- DT connectors
  - arduino_i2c: &i2c1 {}
Issues and limitations

- `.fixup` files need to be populated manually
- No easy handling of device instantiation code
- DT use for some domains requires more than `#define`'s
  - Pinmux generation
- Kconfig processed before dts
  - Kconfig options might be activated while not possible on available HW
- No possible reuse of available dts information (need additional board yaml files for CI)
- No tool available to ease configuration by the user
#ifdef CONFIG_I2C_4
static const struct i2c_config i2c_dev_cfg_4 = {
    .i2c = (I2C_TypeDef*)CONFIG_I2C_4_BASE_ADDRESS,
#ifdef CONFIG_I2C_STM32_INTERRUPT
    .irq_config_func = irq_config_func_4,
#endif
    .bitrate = CONFIG_I2C_4_BITRATE,
};
static struct i2c_data i2c_dev_data_4;

And so on for other instances ...

#endif /* CONFIG_I2C_4 */

#ifdef CONFIG_I2C_STM32_INTERRUPT
static void irq_config_func_4(struct device *dev)
{
    IRQ_CONNECT(CONFIG_I2C_4_EVENT_IRQ,
                 CONFIG_I2C_4_EVENT_IRQ_PRI,
                 stm32_i2c_event_isr, DEVICE_GET(i2c_stm32_4), 0);
    irq_enable(CONFIG_I2C_4_EVENT_IRQ);
}
#endif

DEVICE_AND_API_INIT(i2c_stm32_4, CONFIG_I2C_4_NAME, &i2c_stm32_init, &i2c_dev_data_4, &i2c_dev_cfg_4, POST_KERNEL, CONFIG_KERNEL_INIT_PRIORITY_DEVICE, &api_funcs);

#endif /* CONFIG_I2C_4 */
Zephyr drivers instantiation code

```c
#ifdef CONFIG_I2C_4
sstatic const struct i2c_config i2c_dev_cfg_4 = {
    .i2c = (I2C_TypeDef*) CONFIG_I2C_4_BASE_ADDRESS,
#ifdef CONFIG_I2C_STM32_INTERRUPT
    .irq_config_func = irq_config_func_4,
#endif
    .bitrate = CONFIG_I2C_4_BITRATE,
};
sstatic struct i2c_data i2c_dev_data_4;

Black: Zephyr Generic
Blue: Driver specific
Red: device tree

#ifdef CONFIG_I2C_STM32_INTERRUPT
static void irq_config_func_4(struct device *dev)
{
    IRQ_CONNECT(CONFIG_I2C_4_EVENT_IRQ, CONFIG_I2C_4_EVENT_IRQ_PRI,
                stm32_i2c_event_isr, DEVICE_GET(i2c_stm32_4), 0);
    irq_enable(CONFIG_I2C_4_EVENT_IRQ);
}
#endif

DEVICE_AND_API_INIT(i2c_stm32_4, CONFIG_I2C_4_NAME, &i2c_stm32_init,
            &i2c_dev_data_4, &i2c_dev_cfg_4, POST_KERNEL,
            CONFIG_KERNEL_INIT_PRIORITY_DEVICE,
            &api_funcs);
#endif /* CONFIG_I2C_4 */
```
Codegen introduction and first learnings

- **Codegen** proposal done by Bobby Noelte
  - Code generation python library
  - Derived from Ned Batchelder’s cog python library
  - Parse and execute **python snippets embedded in (c) files comments**

- Some recommendations on codegen use from initial reviews
  - Python code should be hidden behind an intuitive API to enable use by non python aware users. **API could be provided as codegen module**
  - Codegen usage should be constrained to limit abuses of enthusiastic users:
    - “Don't do anything with the preprocessor that can be done in C.”
    - “Don't do anything with codegen that can be done with the preprocessor.”
    - “Don't do anything with CMake that can be done with codegen.”
  - Requires generation of an **extended device tree database** as input to codegen
DT based instantiation code generation

```
"/soc/spi@40003800": {
  "compatible": {
    "0": "st,stm32-spi-fifo"
  },
  "label": "SPI_2",
  "device-type": {
    0 : "spi"},
  "reg": {
    "0": {
      "size": {
        "0": 1024},
      "address": {
        "0": 1073756160}
    }
  },
  "interrupts": {
    "0": {
      "priority": 3,
      "irq": 26
    }
  }
}[
```

extract_dts_include.py

...
DT based instantiation code generation

drivers/i2c/i2c_ll_stm32.c

/**
 * @endcode
 * codegen.import_module('devicedeclare')
 * 
 * device_data = ('i2c_stm32_data')
 *
 * device_config = ('i2c_stm32_config',
 * ""
 * .i2c = (I2C_TypeDef *)${reg/0/address/0},
 * .pclken = {
 *     .enr = ${clocks/0/bits},
 *     .bus = ${clocks/0/bus},
 * },
 * #ifdef CONFIG_I2C_STM32_INTERRUPT
 *     .irq_config_func = ${device-name}_config_irq,
 * #endif
 *     .bitrate = ${clock-frequency},
 * "",
 * )
 * [...]
DT based instantiation code generation

```python
[...]
*device_declare.device_declare(['st,stm32-i2c-v1','st,stm32-i2c-v2'],
* 'CONFIG_KERNEL_INIT_PRIORITY_DEVICE',
* 'POST_KERNEL',
* {'irq_func': 'stm32_i2c_isr', 'irq_flag': CONFIG_I2C_STM32_INTERRUPT},
* 'i2c_stm32_init',
* 'api_funcs',
* 'device_data',
* 'device_config'
*)
*
*@endcode{.codegen}
*/
```
DT based instantiation code generation

Codegen snippets
- `edts.json`
- `#define's`

Python scripts
- `extract_dts_include.py`
- `CodeGen.py`
- `devicedeclare module`

Output
- `outdir/zephyr/drivers/i2c/i2c_ll_stm32.c`
DT based instantiation code generation

```
#ifdef CONFIG_I2C_1

static const struct i2c_stm32_config st_stm32_i2c_v1_40005400_config = {
    .i2c = (I2C_TypeDef *)1073763328,
    .pclken = {
        .enr = 2097152,
        .bus = 2,
    },
#ifdef CONFIG_I2C_STM32_INTERRUPT
    .irq_config_func = st_stm32_i2c_v1_40005400_config_irq,
#endif
    .bitrate = 400000,
};

static struct i2c_stm32_data st_stm32_i2c_v1_40005400_data = {
};
```

...
DT based instantiation code generation

```c
#ifdef CONFIG_I2C_STM32_INTERRUPT
DEVICE_DECLARE(st_stm32_i2c_v1_40005400);
static void st_stm32_i2c_v1_40005400_config_irq(struct device *dev) {
    IRQ_CONNECT(31, 0, stm32_i2c_isr_event, DEVICE_GET(st_stm32_i2c_v1_40005400), 0);
    irq_enable(31);
    IRQ_CONNECT(32, 0, stm32_i2c_isr_error, DEVICE_GET(st_stm32_i2c_v1_40005400), 0);
    irq_enable(32);
}
#endif /* CONFIG_I2C_STM32_INTERRUPT */
```
DT based instantiation code generation

CodeGen snippet

...edts.json
#define's
extract_dts_include.py

yaml

python dict

[...]

DEVICE_AND_API_INIT( \
    st stm32_i2c_v1_40005400, \
    "I2C_1",
    i2c_stm32_init, \
    &st stm32_i2c_v1_40005400_data, \
    &st stm32_i2c_v1_40005400_config, \
    POST_KERNEL, \
    CONFIG_KERNEL_INIT_PRIORITY_DEVICE, \
    &api_func);

#endif /* CONFIG_I2C_1 */
DT based instantiation code generation

- Codegen snippets
  - `edts.json`
  - `#define's`

- `extract_dts_include.py`

- `.yaml`

- `python dict`

- `Codegen.py` + `devicedeclare module`

- `outdir/zephyr/drivers/...`
DT based instantiation code generation

CodeGen snippets

**edts.json**

#define’s

extract_dts_include.py

CodeGen.py + devicedeclare module

outdir/zephyr/drivers/...

CodeGen or other

**edts bonus:** Any piece of board data configuration useful for CI, sanity checks, ...
codegen related open points

● Solution acceptable from the users community?
  ○ [x] Do we keep python snippets (acceptable for non python aware ones)
  ○ [ ] Move to more C-like macro API

● Which file to generate?
  ○ [x] Generate whole c file with instantiation code as add on
  ○ [ ] Generate separate header file
DT generation based open points

- **DTS metadata should be made available to Kconfig (GH#7302)**
  - Run dt generation first: HW configuration
  - Then control generated code compilation with Kconfig: SW configuration

- **Configuration flags generation from DT**
  - Generation of CONFIG_ flags creates a confusion on where the configuration is done
  - Introducing a DTS_ namespace is required
  - Need to make DTS_ flags visible to Kconfig. Specify how they interact (x-dependencies, ..)

- **DT and driver files build control (GH#9406)**
  - Generate DTS_<COMPATIBLE> flags based on ‘compatible’
  - `zephyr_library_sources_ifdef(DTS_STM32_I2C_V1 i2c_ll_stm32_v1.c)`
  - simple and efficient ‘HW’ <> ‘SW’ binding

- **UI for dts configuration (dtsconfig)**
  - Needed?
Device Tree needs for TFM

- How do we describe secure v non-secure
- Use of ranges
  - Do we need dma-ranges?
- Tooling for validation and generation
  - Catch Errors from mis-configuration of secure v non-secure partitioning of resources
  - Generation tools from DT to secure configuration
DT Miscellaneous Issues

- DTC warnings with Zephyr (unit-address, etc)
- Possible bug in DTC w/phandle to non-unit-address node
- Fixes to Zephyr generation script for various use cases/patterns we don’t normally see