Zephyr® Project
Developer Summit

Configure Zephyr: Kconfigs and Devicetree in Simple Words

Roy Jamil, Ac6

About Me

- Training engineer at Ac6 specialized in RTOS and Linux
- PhD from ENSMA – France
- Authored a comprehensive Zephyr training course
- Provide consulting and support on Zephyr and Linux

About Ac6

- Specialized in embedded systems training and support for over 20 years
- IDE development : SW4STM32
- Training catalog with more than 150 courses
- Focus areas include RTOS, Linux, Security, FPGA and more
- Zephyr training covers basics to advanced
  - Configuring Zephyr
  - Device Driver Model
  - Common Subsystems

Purpose

- Simplify adopting Zephyr
  - A big challenge to adopt it is due to unfamiliarity with Kconfigs and Devicetree

- Clarify their complexity misconceptions
  - Showcase their simplicity and utility

- Provide a clear understanding of how Kconfigs and Devicetree enhance the Zephyr development experience

Overview

2 Configuration Frameworks

Kconfig

- System configuration
  - Enable or disable global features
- Conditional compilation
  - In C code or CMake
- Set default values
- Kernel tuning
- Options visualization
  - menuconfig/guiconfig

Devicetree

- Hardware description
  - Details about devices
  - Peripheral Configuration
  - Memory mappings
  - Interrupt lines
- Platform agnostic
  - Facilitates firmware portability across different hardware platforms by abstracting hardware-specific details

Build-time configuration

2 Roles

Application developer

Customize your specific application

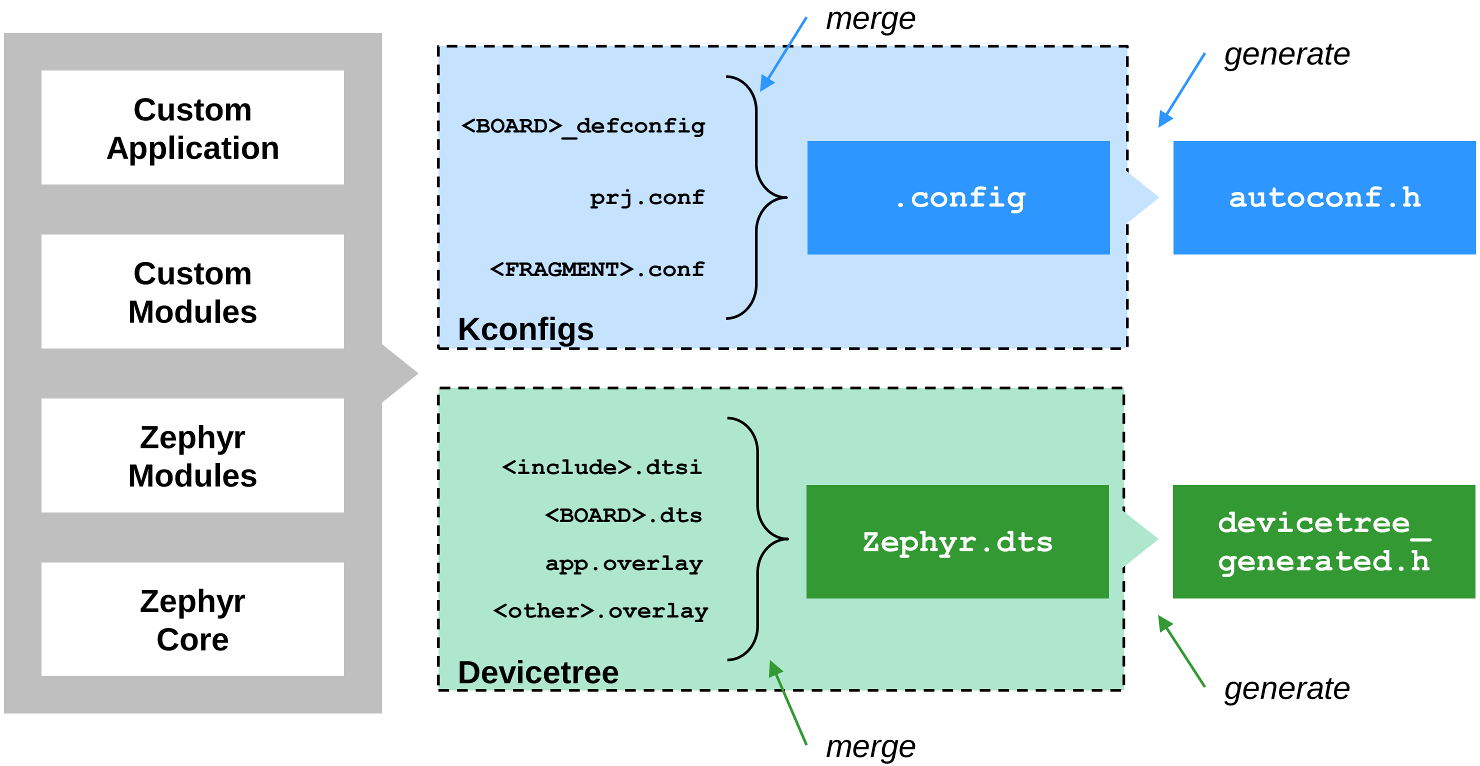
- Configure the environment to suit the needs of a particular project
- Set the values in one of the used files
  - One standard file per project
    - `prj.conf` and `app.overlay`
  - Custom extra overlay/fragment files

Platform developer

Extend Zephyr's Capabilities

- Introduce new options
  - When adding support for new features, drivers, or modules
- Define new configuration parameters
  - Users can later set according to their project requirements
- Set the values in one of the used files
  - One default file per board
  - Some other files

Overview



EMBEDDED OPEN SOURCE SUMMIT

Examples

Kconfig

- In a conf file:

```
CONFIG_SERIAL=y
CONFIG_UART_MCUX_LPUART=y
```

- In CMake:

```
zephyr_library_sources_ifdef(
   CONFIG_UART_MCUX_LPUART
   uart_mcux_lpuart.c)
```

- In source code:

```
#ifdef CONFIG_UART_MCUX_LPUART
/* Some code */
#endif /* CONFIG_UART_MCUX_LPUART */
```

Devicetree

- In devicetree source or overlay:

```
flexcomm4_lpuart4: lpuart@b4000 {
   compatible = "nxp,kinetis-lpuart";
   reg = <0xb4000 0x1000>;
   current-speed = <115200>;
   status = "okay";
};
```

- In source code:

```
DT_INST_PROP(idx, current_speed)
```

Kconfig

Kconfig options

- **They can be used to :**
  - Enable or disable specific features in the application
  - Define default values for configuration options
  - Set boundaries, such as minimum or maximum possible values
  - Configure protocols
  - Fine-tune the kernel and scheduler
  - Enable complex conditional configurations without the need for manual adjustments

- **Limitations**
  - They are not suitable for configuring specific devices or declaring device instances
  - Designed for global configuration settings rather than fine-grained device-specific settings

Kconfig advantages

- **Flexibility:**
  - Customize the project in a convenient way with the desired functionalities
- **Modularity:**
  - Modular code organization by enabling/disabling needed features
- **Simplify Configuration Management:**
  - A centralized and structured approach to manage project configurations.
- **Consistency:**
  - Enforce a consistent configuration approach across different projects

Initial configuration file

- The application must be configured before being built
- The final configuration is stored in **`build/zephyr/.config`**
- The initial .config is generated from merging several files:
  1. The default config:
     - **`<BOARD>_defconfig`** (e.g. **`frdm_mcxn947_cpu0_defconfig`**)
  2. prj.conf
     - Only one file per project
  3. Extra config fragment
     - Any file listed in this variable: **`EXTRA_CONF_FILE`**

Note: All these files use the same syntax

Application developer's role

- The **vendor** will provide a defconfig that has some options enabled by default
  - Minimum necessary settings

- The **developer** usually needs to enable additional features

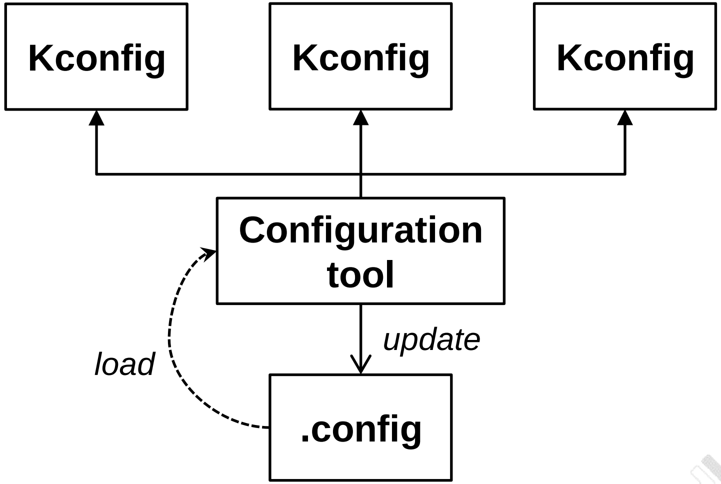
- The `.config` can be modified temporarily using an interactive tool
  - Modifications will be discarded after deleting the build directory (pristine build)
    - `$ west build -t pristine`

- To make the modification persistent, you should place your options either in prj.conf or in an extra configuration fragment
  - Syntax: `CONFIG_<symbol name>=<value>` (e.g. `CONFIG_GPIO=y`)

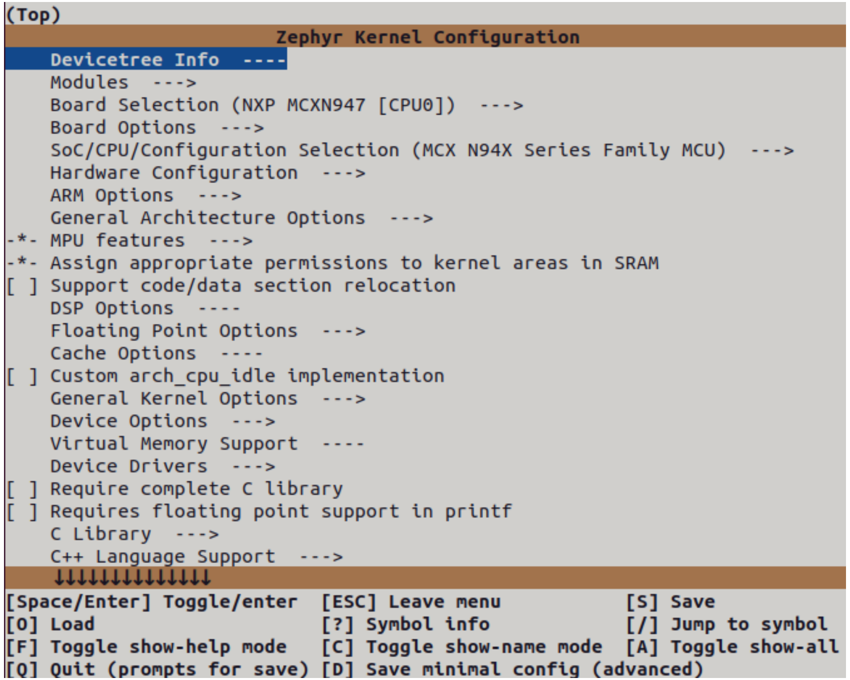
Interactive Kconfig configuration tools

- It is recommended to set options using these tools

- Advantages include:
  - Not allow enabling/disabling an option if there are dependencies to/on it
  - User-Friendly interface
  - Search
  - Help
  - See related options
  - Automatically create `.config.old` backup

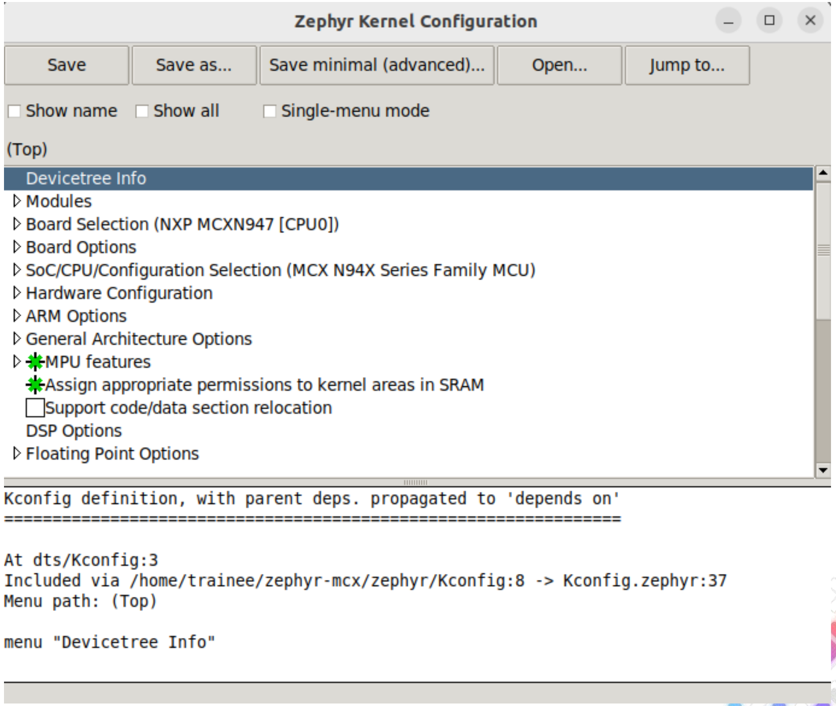


menuconfig



```
(Top)
                          Zephyr Kernel Configuration
    Devicetree Info  ----
    Modules  --->
    Board Selection (NXP MCXN947 [CPU0])  --->
    Board Options  --->
    SoC/CPU/Configuration Selection (MCX N94X Series Family MCU)  --->
    Hardware Configuration  --->
    ARM Options  --->
    General Architecture Options  --->
-*- MPU features  --->
-*- Assign appropriate permissions to kernel areas in SRAM
[ ] Support code/data section relocation
    DSP Options  ----
    Floating Point Options  --->
    Cache Options  ----
[ ] Custom arch_cpu_idle implementation
    General Kernel Options  --->
    Device Options  --->
    Virtual Memory Support  ----
    Device Drivers  --->
[ ] Require complete C library
[ ] Requires floating point support in printf
    C Library  --->
    C++ Language Support  --->
    ↓↓↓↓↓↓↓↓↓↓↓↓↓↓
[Space/Enter] Toggle/enter   [ESC] Leave menu           [S] Save
[O] Load                     [?] Symbol info            [/] Jump to symbol
[F] Toggle show-help mode    [C] Toggle show-name mode  [A] Toggle show-all
[Q] Quit (prompts for save)  [D] Save minimal config (advanced)
```

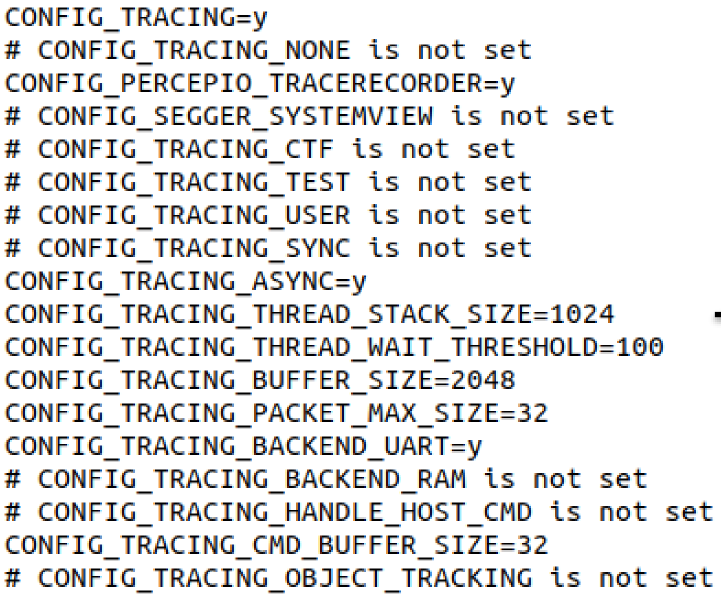
guiconfig



```
Devicetree Info
▷ Modules
▷ Board Selection (NXP MCXN947 [CPU0])
▷ Board Options
▷ SoC/CPU/Configuration Selection (MCX N94X Series Family MCU)
▷ Hardware Configuration
▷ ARM Options
▷ General Architecture Options
▷ ✱MPU features
  ✱Assign appropriate permissions to kernel areas in SRAM
  □Support code/data section relocation
  DSP Options
▷ Floating Point Options
```

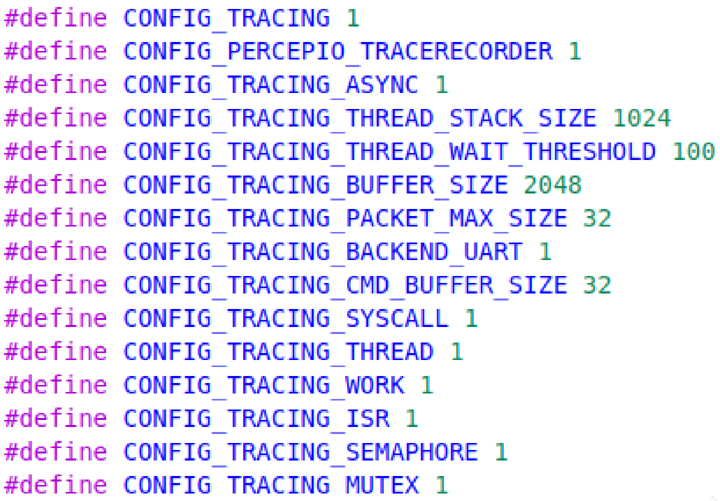
```
Kconfig definition, with parent deps. propagated to 'depends on'
=================================================================

At dts/Kconfig:3
Included via /home/trainee/zephyr-mcx/zephyr/Kconfig:8 -> Kconfig.zephyr:37
Menu path: (Top)

menu "Devicetree Info"
```

.config



```
CONFIG_TRACING=y
# CONFIG_TRACING_NONE is not set
CONFIG_PERCEPIO_TRACERECORDER=y
# CONFIG_SEGGER_SYSTEMVIEW is not set
# CONFIG_TRACING_CTF is not set
# CONFIG_TRACING_TEST is not set
# CONFIG_TRACING_USER is not set
# CONFIG_TRACING_SYNC is not set
CONFIG_TRACING_ASYNC=y
CONFIG_TRACING_THREAD_STACK_SIZE=1024
CONFIG_TRACING_THREAD_WAIT_THRESHOLD=100
CONFIG_TRACING_BUFFER_SIZE=2048
CONFIG_TRACING_PACKET_MAX_SIZE=32
CONFIG_TRACING_BACKEND_UART=y
# CONFIG_TRACING_BACKEND_RAM is not set
# CONFIG_TRACING_HANDLE_HOST_CMD is not set
CONFIG_TRACING_CMD_BUFFER_SIZE=32
# CONFIG_TRACING_OBJECT_TRACKING is not set
```

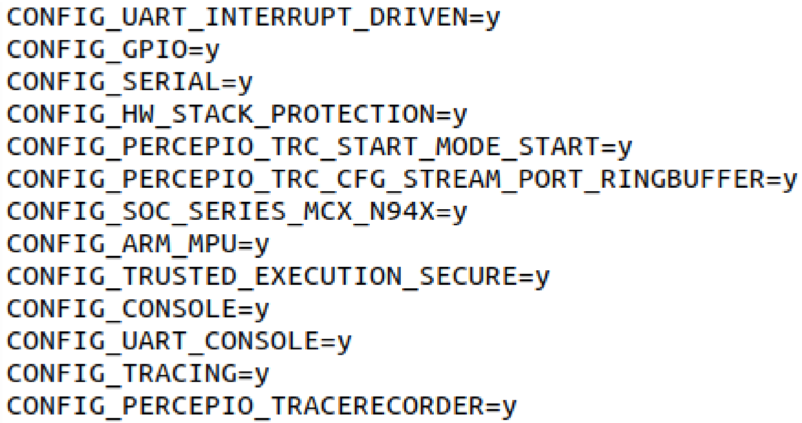
autoconf.h



```
#define CONFIG_TRACING 1
#define CONFIG_PERCEPIO_TRACERECORDER 1
#define CONFIG_TRACING_ASYNC 1
#define CONFIG_TRACING_THREAD_STACK_SIZE 1024
#define CONFIG_TRACING_THREAD_WAIT_THRESHOLD 100
#define CONFIG_TRACING_BUFFER_SIZE 2048
#define CONFIG_TRACING_PACKET_MAX_SIZE 32
#define CONFIG_TRACING_BACKEND_UART 1
#define CONFIG_TRACING_CMD_BUFFER_SIZE 32
#define CONFIG_TRACING_SYSCALL 1
#define CONFIG_TRACING_THREAD 1
#define CONFIG_TRACING_WORK 1
#define CONFIG_TRACING_ISR 1
#define CONFIG_TRACING_SEMAPHORE 1
#define CONFIG_TRACING_MUTEX 1
```

Save minimal config

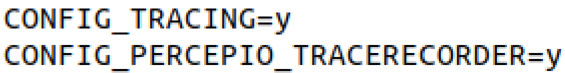
- An option within the configuration tools
- Generates a clean and minimal defconfig
- Only settings that differ from the default values are included in the defconfig
- Serves as a base for new configurations or variations
- Ideal for creating config fragments
    - Previously called overlay config



```
CONFIG_UART_INTERRUPT_DRIVEN=y
CONFIG_GPIO=y
CONFIG_SERIAL=y
CONFIG_HW_STACK_PROTECTION=y
CONFIG_PERCEPIO_TRC_START_MODE_START=y
CONFIG_PERCEPIO_TRC_CFG_STREAM_PORT_RINGBUFFER=y
CONFIG_SOC_SERIES_MCX_N94X=y
CONFIG_ARM_MPU=y
CONFIG_TRUSTED_EXECUTION_SECURE=y
CONFIG_CONSOLE=y
CONFIG_UART_CONSOLE=y
CONFIG_TRACING=y
CONFIG_PERCEPIO_TRACERECORDER=y
```

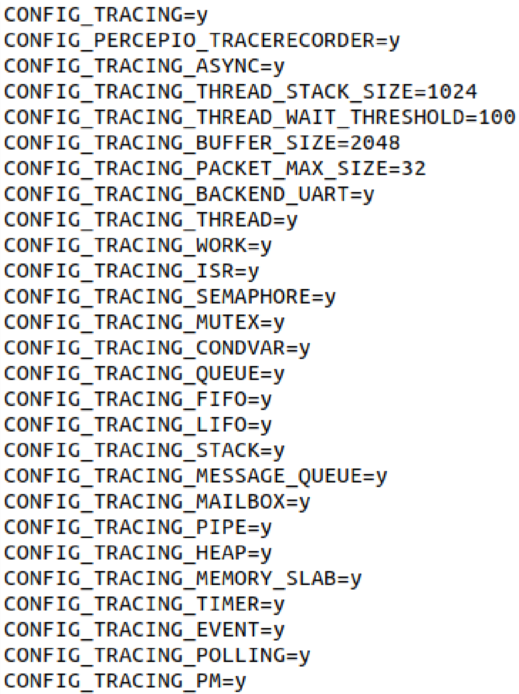
NOTE: The `.config` for the same project is 1167 lines

Minimal



```
CONFIG_TRACING=y
CONFIG_PERCEPIO_TRACERECORDER=y
```

Not Minimal



```
CONFIG_TRACING=y
CONFIG_PERCEPIO_TRACERECORDER=y
CONFIG_TRACING_ASYNC=y
CONFIG_TRACING_THREAD_STACK_SIZE=1024
CONFIG_TRACING_THREAD_WAIT_THRESHOLD=100
CONFIG_TRACING_BUFFER_SIZE=2048
CONFIG_TRACING_PACKET_MAX_SIZE=32
CONFIG_TRACING_BACKEND_UART=y
CONFIG_TRACING_THREAD=y
CONFIG_TRACING_WORK=y
CONFIG_TRACING_ISR=y
CONFIG_TRACING_SEMAPHORE=y
CONFIG_TRACING_MUTEX=y
CONFIG_TRACING_CONDVAR=y
CONFIG_TRACING_QUEUE=y
CONFIG_TRACING_FIFO=y
CONFIG_TRACING_LIFO=y
CONFIG_TRACING_STACK=y
CONFIG_TRACING_MESSAGE_QUEUE=y
CONFIG_TRACING_MAILBOX=y
CONFIG_TRACING_PIPE=y
CONFIG_TRACING_HEAP=y
CONFIG_TRACING_MEMORY_SLAB=y
CONFIG_TRACING_TIMER=y
CONFIG_TRACING_EVENT=y
CONFIG_TRACING_POLLING=y
CONFIG_TRACING_PM=y
```

Permanent config

- To create a permanent config:
  - **prj.conf**
    - Only one file per project
  - **config fragment** (overlay)
    - Put the options you want to set into a file
      - Either write them directly or use the configuration tools
    - In the main CMakeLists.txt : `set(EXTRA_CONF_FILE path/to/my_config1.conf)`
      - **NOTE**: it should be added before `find_package(Zephyr)`

Config fragment

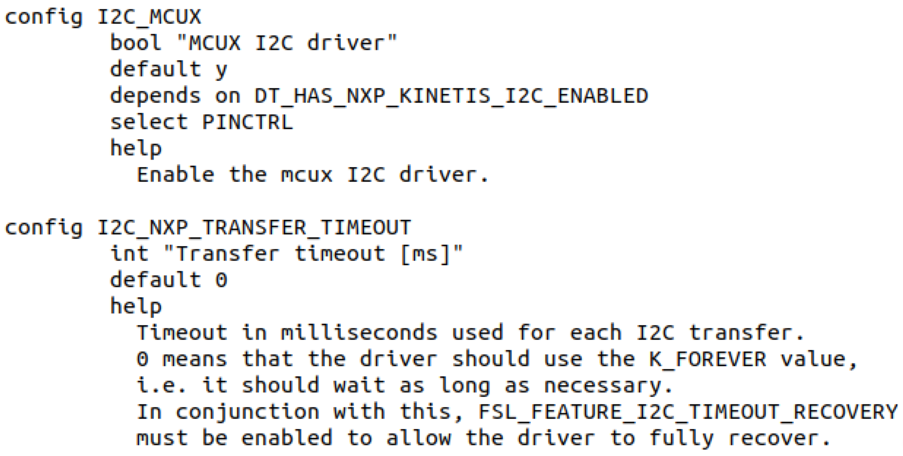
- Using the minimal config:
  - In menu/guiconfig: Save minimal the original config as a backup
  - Enable and modify the options you want
  - Then save the updated config in a new file
  - Compare both files (manually or using `diff`), for example :

```
$ diff --changed-group-format='%>' --unchanged-group-format='' defconf.orig defconf.new > my.conf
```

- Using the not minimal config:
  - Enable and modify the options you want using menu/guiconfig
  - Save (not minimal)
    - A backup file will be automatically created in `build/zephyr/.config.old`
  - Compare `.config.old` and `.config` same as the previous method

Platform developer’s role

- Extend Zephyr Kconfig by adding new custom Kconfigs
- Define new options (symbols)
- Integrate features into the source code and CMake files



```
config I2C_MCUX
        bool "MCUX I2C driver"
        default y
        depends on DT_HAS_NXP_KINETIS_I2C_ENABLED
        select PINCTRL
        help
          Enable the mcux I2C driver.

config I2C_NXP_TRANSFER_TIMEOUT
        int "Transfer timeout [ms]"
        default 0
        help
          Timeout in milliseconds used for each I2C transfer.
          0 means that the driver should use the K_FOREVER value,
          i.e. it should wait as long as necessary.
          In conjunction with this, FSL_FEATURE_I2C_TIMEOUT_RECOVERY
          must be enabled to allow the driver to fully recover.
```

EMBEDDED
OPEN SOURCE
SUMMIT

Devicetree

Devicetree

- Single source for hardware information
  - Device drivers obtain configurable hardware descriptions from the devicetree.
  - New device drivers use devicetree APIs to create devices based on hardware configurations.

- Advantages of devicetree in Zephyr:
  - Configurability:
    - Devicetree enables hardware descriptions to be easily configurable
    - No need to change C source code to reconfigure devices
  - Proven concept:
    - A standardized format used by other projects like Linux, u-boot, ATF…

Syntax

- The devicetree describes the platform as a tree
  - root is called ‘/’
- A tree is made of nodes
  - nodes are defined between ‘{’ and ‘}’
- Nodes are named in the following pattern :

```
[node-label:] node-name[@unit-address]
```

  - **node-name**: generic name, reflecting the function of the device
  - **unit-address**: [optional], must match the address specified in the reg property
  - **label**: [optional], name used to identify a node and to make references to it
  - Example: **timer1: timer@4a318000**
- Each node has properties, they look like C assignments and end with a ‘;’

Property types and values

- Integer (up to 32 bits)
  - `prop = < 16 >;` or `prop = < 0x10 >;`
- Phandle (reference to a node)
  - `prop = <&node-label>;`
- Integer or references array
  - `prop = < 1 2 &node-label >;`
- String
  - `prop = "hello";`
- String list
  - `prop = "hello", "world";`
- No values
  - `prop;`
    - The presence or the absence of the property can be seen as its boolean value

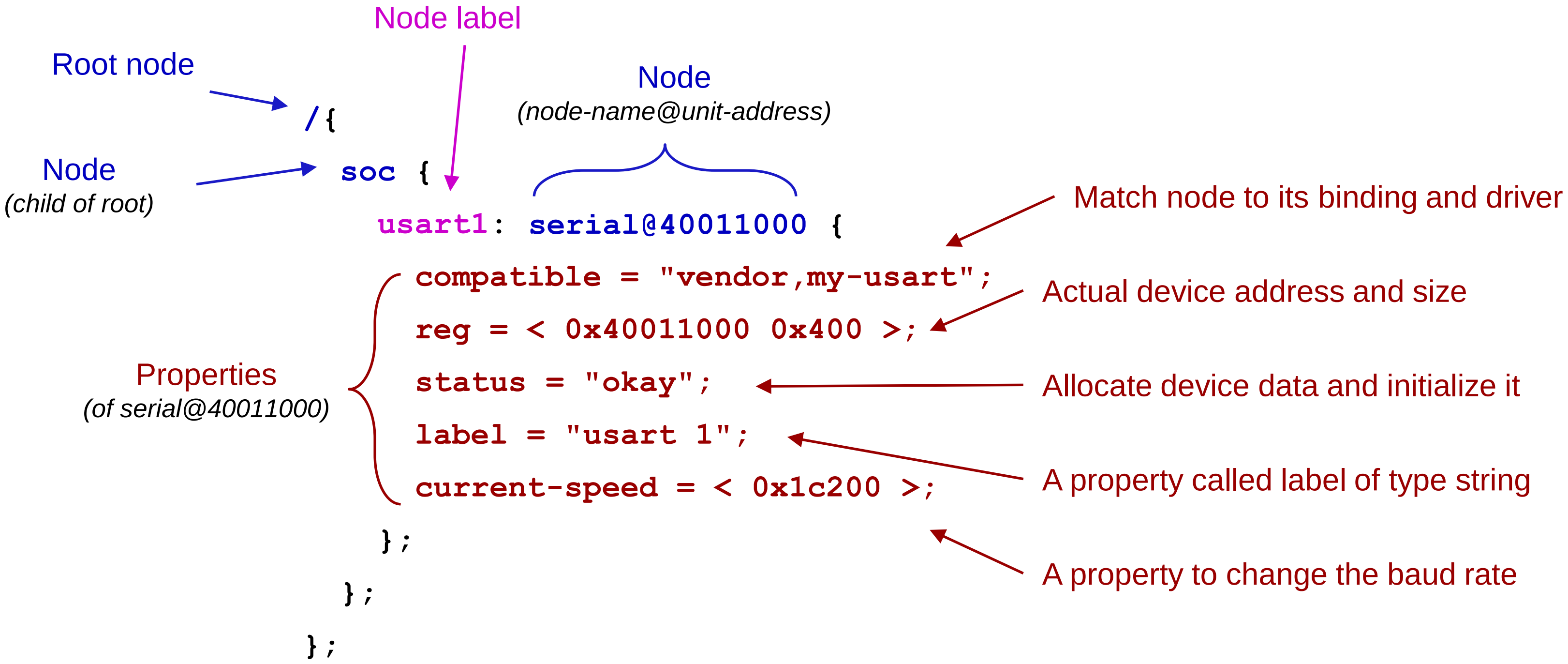
Standard properties

- **`compatible`** (string list)
  - Used to match devicetree nodes with both the source code of the driver and bindings (yaml)

- **`status`** (string)
  - The valid values are : "`okay`", "`disabled`"
  - Allocate and initialize the device when "okay"

- **`reg = <address size ...>`** (u32 cells)
  - Provides the base address and the size of a node
  - **`#address-cells`**: defines the number of <u32> cells used to encode the address field
  - **`#size-cells`**: defines the number of <u32> cells used to encode the size field

Example: Devicetree



```
/{
  soc {
    usart1: serial@40011000 {
      compatible = "vendor,my-usart";
      reg = < 0x40011000 0x400 >;
      status = "okay";
      label = "usart 1";
      current-speed = < 0x1c200 >;
    };
  };
};
```

Node label
Root node
Node
(node-name@unit-address)
Node
(child of root)
Properties
(of serial@40011000)
Match node to its binding and driver
Actual device address and size
Allocate device data and initialize it
A property called label of type string
A property to change the baud rate

Initial devicetree file

- The devicetree is generated from merging several files:
  - The main device tree source (DTS):
    - `<BOARD>.dts` (e.g. `frdm_mcxn947_cpu0.dts`)
  - `app.overlay`
    - Only one file per project
  - Extra devicetree overlays
    - Any file listed in this variable: `EXTRA_DTC_OVERLAY_FILE`

- The final devicetree is stored in `build/zephyr/zephyr.dts`

- Note: All these files use the same syntax

#include

- Using the C preprocessor `#include`, it is possible include other files
    - .dtsi files
        - Common configurations with a family of boards and SoCs
        - Default configuration for supported devices
    - C header files (.h)
        - Define hardware constants instead of directly providing raw values
            - E.g: **`ADC0_A1_PIO4_15`**
        - Parameters that influence the behavior/initialization of the driver
            - E.g: **`GPIO_ACTIVE_HIGH`**

- A default configuration is usually provided for all supported peripherals
    - Most of them are disabled
    - The **`<BOARD>.dts`** will enable and reconfigure the used peripherals

Application developer’s role

- Any developer will need to customize the default devicetree
  - Usually done in an overlay file
    - app.overlay
    - Adding an extra overlay

- Use devicetree data in source code
  - Retrieve the device pointer (struct device *)
    - Needed for performing operations on the specific device
  - Retrieve property values
    - Accessing various hardware configuration parameters

Example: devicetree_generated.h

```
#define DT_N_S_soc_S_serial_40011000_P_compatible {"vendor,my-usart"}
#define DT_N_S_soc_S_serial_40011000_P_compatible_LEN 1
#define DT_N_S_soc_S_serial_40011000_P_compatible_EXISTS 1
#define DT_N_S_soc_S_serial_40011000_P_current_speed 115200
#define DT_N_S_soc_S_serial_40011000_P_status "okay"
#define DT_N_S_soc_S_serial_40011000_P_label "usart 1"
#define DT_N_S_soc_S_serial_40011000_P_reg {1073811456 /* 0x40011000 */, \
                                            1024 /* 0x400 */}
```

Access devicetree from source code

- Retrieve the node id
  - `DT_PATH(path, to, node)`
  - `DT_NODELABEL(node_label)`
  - Example: `#define USART1_NODE DT_NODELABEL(usart1)`

- Get a property value
  - `DT_PROP(node_id, prop)`
  - Example: `uint32_t baud_rate = DT_PROP(USART1_NODE, current_speed)`
    - Note: you should replace the ‘-’ in devicetree by ‘_’ in source code

- Some drivers have specific macros, example:
  - `const struct gpio_dt_spec my_led = GPIO_DT_SPEC_GET(LED_NODE, gpios);`

struct device *

- All devices should have an instance of type struct device
  - Zephyr device driver model
  - They are typically defined in the devicetree

- Common methos to retrieve a device reference:

```
const struct device *device_get_binding(const char *name)
```

  - It can be used for devices defined within or without a devicetree

```
DEVICE_DT_GET(node_id)
```

  - Get a device reference from a devicetree node identifier

Example

```
#define I2C_2_NODE    DT_NODELABEL(flexcomm2_lpi2c2)

const struct device *i2c_dev = (const struct device *) DEVICE_DT_GET(I2C_2_NODE);

if (!device_is_ready(i2c_dev)) {
   printk("I2C device not ready.\n");
   return ERROR;
}

if (i2c_write(i2c_dev, data, sizeof(data), ADDR)) {
   printk("Failed to write.\n");
   return ERROR;
}
```

Platform developer's role

- Extending Zephyr hardware support
  - Adding new drivers
  - Porting boards and drivers to Zephyr

- Drivers must respect the Zephyr device driver model

- Typically involves adding new devicetree elements
  - The driver should automatically define devices for each compatible node with status "okay"
    - This is done thanks to **`DT_FOREACH_STATUS_OKAY_NODE(fn)`**

Bindings

- Why ?
  - Bindings provide the types of the properties used
  - They declare requirements and provide semantic information about valid nodes

- Binding are YAML files
  - The name of the file should match the compatible node
  - Each file should declare the requirements for each property in that node

- Example:

```
properties:
   modem-mode:
     type: int
     required: true
     description: Set the UART Port to modem mode 0 (dce) 64 (dte)
```

Zephyr® Project

Developer Summit