MAB LABS

Embedded Solutions

Zephyr Device Drivers

Walkthrough and Examples

MOHAMMED BILLOO

Embedded World 2024

Agenda

- Code Organization
- Device Driver Model
- Common API
- Relevant Data Structures
- Examples
  - Filesystem
  - SD card (SPI)
- Out-of-tree Drivers

THE SPEAKER

- Embedded Software Consultant
- Design Work
  - Medical devices
  - Scientific instruments
  - LIDAR
  - Custom ASIC
  - Consumer electronics
- Experience/Expertise
  - RTOS-based systems
  - Embedded Linux/The Yocto Project
  - Qt

Mohammed Billoo
(mab@mab-labs.com)

/mab-embedded

@mabembedded

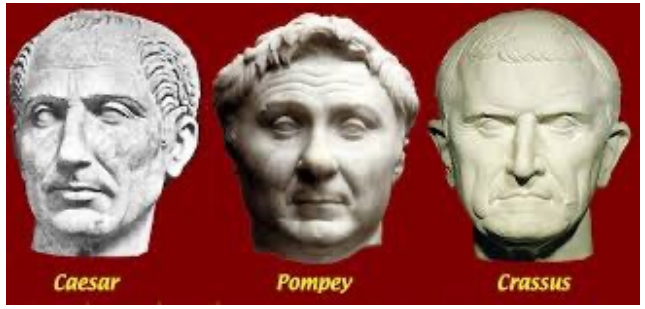
THE SPEAKER

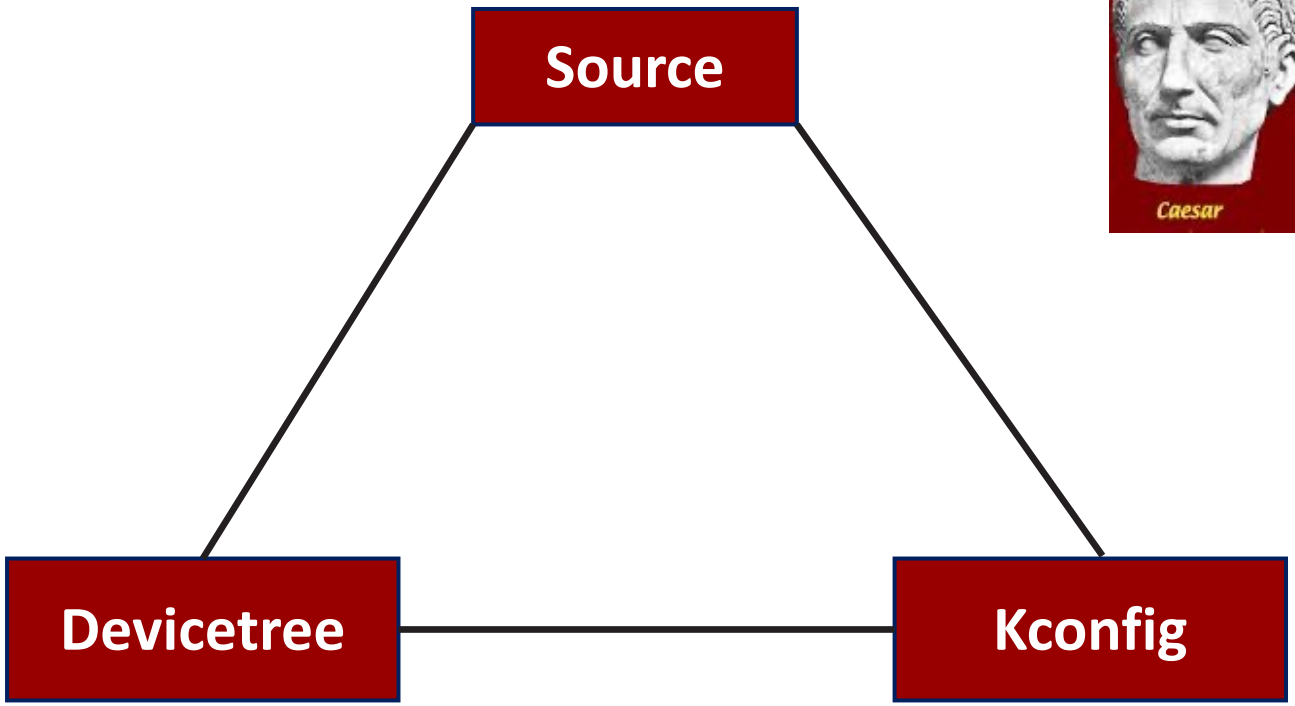
- Training/Workshops
  - Virtual
  - On-site/In-person

BIOS FOOD Newsletter

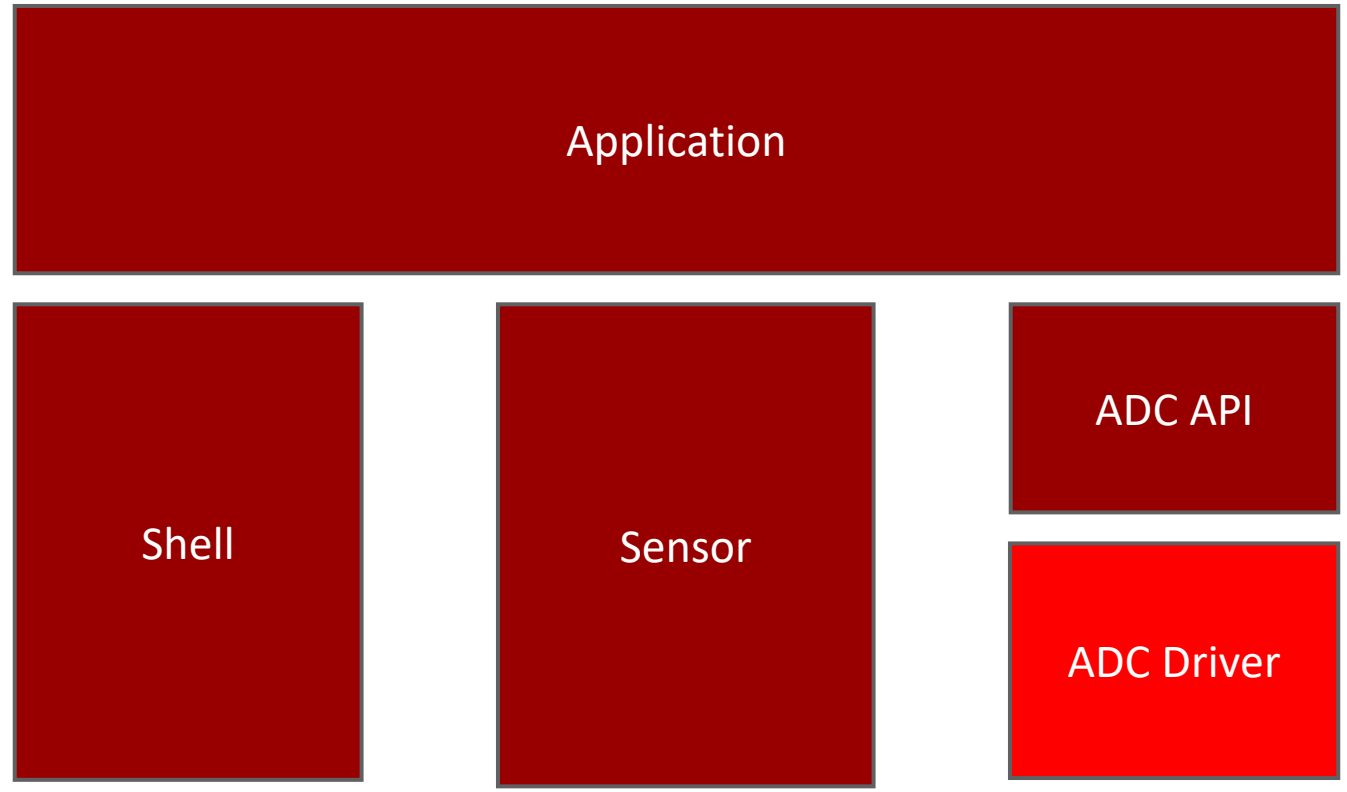
www.mab-labs.com

Zephyr Triumvirate





Zephyr Code Organization



Zephyr

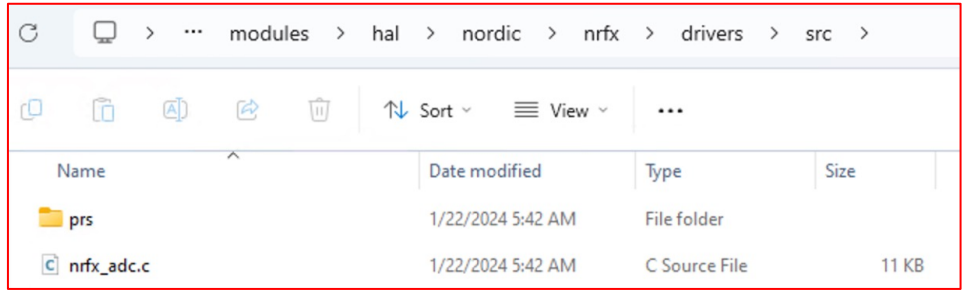
Non-Zephyr

Zephyr Code Organization

```
[zp]$ ls
bootloader  modules  tools  zephyr
```

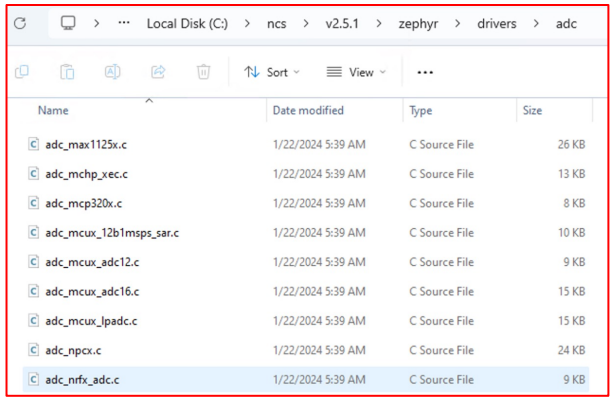
```
[zephyr]$ ls
arch                CODEOWNERS          include          LICENSE           samples      soc           version.h.in
boards              CONTRIBUTING.rst    Kconfig          MAINTAINERS.yml   scripts      submanifests  west.yml
cmake               doc                 Kconfig.zephyr   misc              SDK_VERSION  subsys        zephyr-env.cmd
CMakeLists.txt      drivers             kernel           modules           share        tests         zephyr-env.sh
CODE_OF_CONDUCT.md  dts                 lib              README.rst        snippets     VERSION
```

Example: Nordic ADC



NORDIC SEMICONDUCTOR

Non-Zephyr




```
197     channel_id = 0U;
198     while (selected_channels) {
199         if (selected_channels & BIT(0)) {
200             /* The nrfx driver requires setting the resolution
201              * for each enabled channel individually.
202              */
203             m_channels[channel_id].config.resolution =
204                 nrf_resolution;
205             nrfx_adc_channel_enable(&m_channels[channel_id]);
206             ++active_channels;
207         }
208         selected_channels >>= 1;
209         ++channel_id;
210     }
211
212     error = check_buffer_size(sequence, active_channels);
213     if (error) {
214         return error;
215     }
```

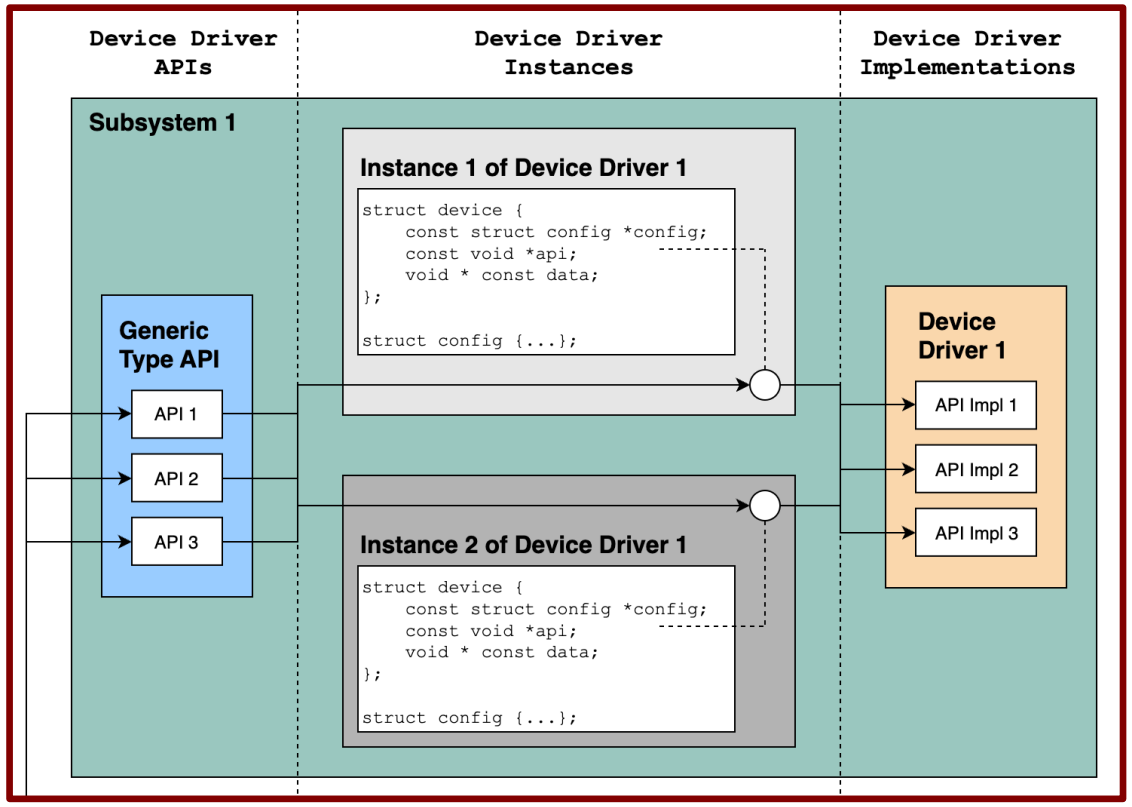
```
284  static const struct adc_driver_api adc_nrfx_driver_api = {
285      .channel_setup = adc_nrfx_channel_setup,
286      .read          = adc_nrfx_read,
287  #ifdef CONFIG_ADC_ASYNC
288      .read_async    = adc_nrfx_read_async,
289  #endif
290      .ref_internal  = 1200,
291  };
292
293  /*
294   * There is only one instance on supported SoCs, so inst is guaranteed
295   * to be 0 if any instance is okay. (We use adc_0 above, so the driver
296   * is relying on the numeric instance value in a way that happens to
297   * be safe.)
298   *
299   * Just in case that assumption becomes invalid in the future, we use
300   * a BUILD_ASSERT().
301   */
302  #define ADC_INIT(inst)                          \
303      BUILD_ASSERT((inst) == 0,                   \
304                   "multiple instances not supported");       \
305      DEVICE_DT_INST_DEFINE(0,                    \
306                    init_adc, NULL, NULL, NULL,          \
307                    POST_KERNEL,                \
308                    CONFIG_ADC_INIT_PRIORITY,            \
309                    &adc_nrfx_driver_api);
310
311  DT_INST_FOREACH_STATUS_OKAY(ADC_INIT)
312
```

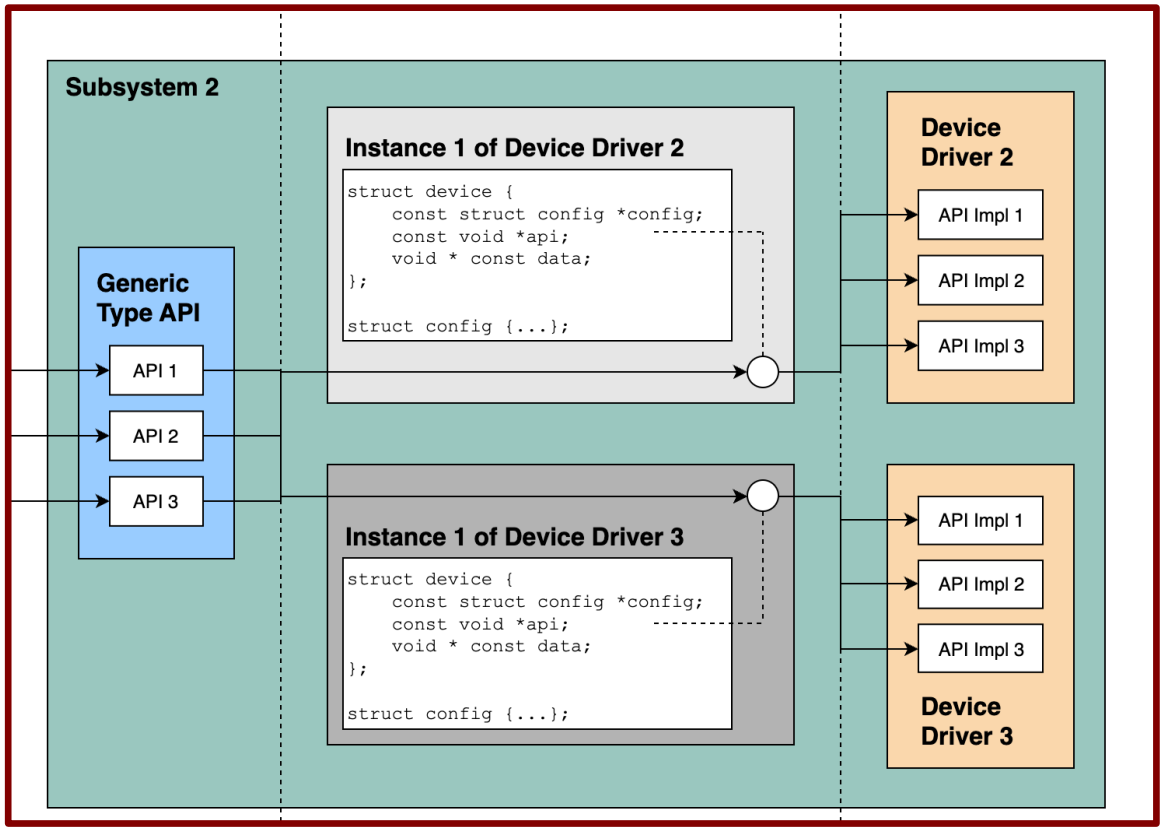
Zephyr

Device Driver Model

- Consistent driver configuration
  - Mostly
- Responsible for initializing the drivers that are part of the system
- Generic type API
  - ADC
  - EEPROM
  - I2C
  - SPI
  - UART

Device Driver Model





Standard Drivers

- Few drivers that are present on all supported boards
  - Inherently necessary to run core kernel operations
- Interrupt controller
  - Needed to kernel to use interrupts
- Timer
  - Needed for kernel to maintain time for system time
- Serial comms
  - Console interface
- “Entropy”
  - PRNG

Common Device API

- DEVICE_DEFINE(dev_id, name, init_fn, pm, data, config, level, prio, api)
  - Creates an instance of the device
  - **You shouldn't use this**
  - Instead use DEVICE_DT_INST_DEFINE
    - Only instantiates device if present in devicetree (i.e. part of hardware system)
    - Acts as devicetree validation (if the device is not part of devicetree, we will get all sorts of macro errors)

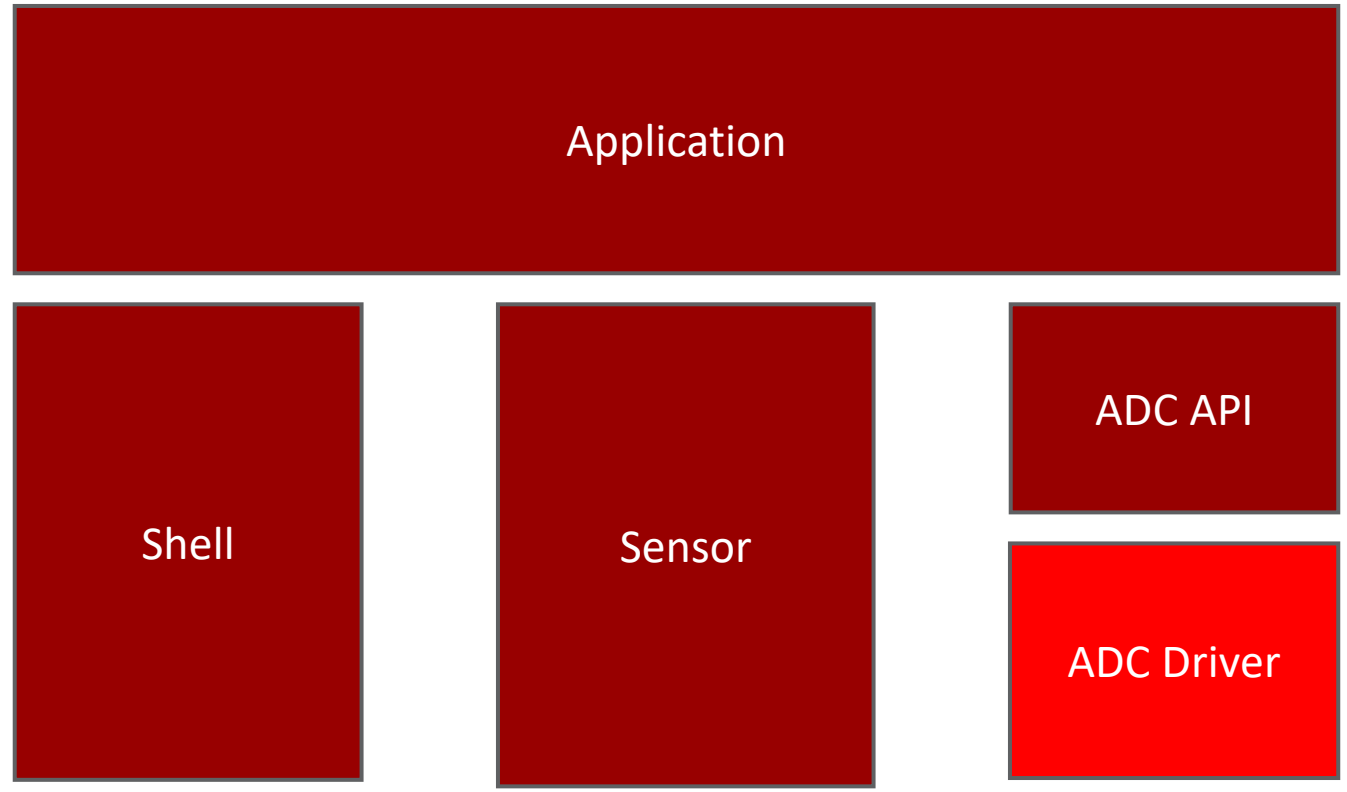
Common Device API

- DEVICE_NAME_GET()
  - Returns ”global device object” representing device
    - Essentially, a C struct
    - Relevant inside driver implementation

Getting Started

```
/**
 * @brief Runtime device structure (in ROM) per driver instance
 */
struct device {
        /** Name of the device instance */
        const char *name;
        /** Address of device instance config information */
        const void *config;
        /** Address of the API structure exposed by the device instance */
        const void *api;
        /** Address of the common device state */
        struct device_state *state;
        /** Address of the device instance private data */
        void *data;
```

Generic Subsystem API



Zephyr

Non-Zephyr

Generic Subsystem API

- Driver must implement API required by subsystem
  - Subsystem calls corresponding function call
  - Ultimately called by the application
- Generic example
  - Available at Zephyr documentation
  - https://docs.zephyrproject.org/latest/kernel/drivers/index.html

Generic Subsystem API

```
typedef int (*subsystem_do_this_t)(const struct device *dev, int foo, int bar);
typedef void (*subsystem_do_that_t)(const struct device *dev, void *baz);

struct subsystem_api {
	subsystem_do_this_t do_this;
	subsystem_do_that_t do_that;
};

static inline int subsystem_do_this(const struct device *dev, int foo, int bar)
{
	struct subsystem_api *api;

	api = (struct subsystem_api *)dev->api;
	return api->do_this(dev, foo, bar);
}

static inline void subsystem_do_that(const struct device *dev, void *baz)
{
	struct subsystem_api *api;

	api = (struct subsystem_api *)dev->api;
	api->do_that(dev, baz);
}
```

Generic Subsystem API

```
static int my_driver_do_this(const struct device *dev, int foo, int bar)
{
        ...
}

static void my_driver_do_that(const struct device *dev, void *baz)
{
        ...
}

static struct subsystem_api my_driver_api_funcs = {
        .do_this = my_driver_do_this,
        .do_that = my_driver_do_that
};
```

Set to “api” argument in “struct device” data structure. “do_this” and “do_that” ultimately called from application,

Generic Subsystem API

- It doesn't prevent the driver from implementing functions outside of the subsystem API

```
static int generic_do_this(const struct device *dev, void *arg)
{
   ...
}

static struct generic_api api {
   ...
   .do_this = generic_do_this,
   ...
};

/* supervisor-only API is globally visible */
int specific_do_that(const struct device *dev, int foo)
{
   ...
}
```

Fat Filesystem Example

subsys/fs/fat_fs.c

```
/* File system interface */
static const struct fs_file_system_t fatfs_fs = {
        .open = fatfs_open,
        .close = fatfs_close,
        .read = fatfs_read,
        .write = fatfs_write,
        .lseek = fatfs_seek,
        .tell = fatfs_tell,
        .truncate = fatfs_truncate,
        .sync = fatfs_sync,
        .opendir = fatfs_opendir,
        .readdir = fatfs_readdir,
        .closedir = fatfs_closedir,
        .mount = fatfs_mount,
        .unmount = fatfs_unmount,
        .unlink = fatfs_unlink,
        .rename = fatfs_rename,
        .mkdir = fatfs_mkdir,
        .stat = fatfs_stat,
        .statvfs = fatfs_statvfs,
#if defined(CONFIG_FILE_SYSTEM_MKFS)
        .mkfs = fatfs_mkfs,
#endif
};
```

```
static int fatfs_open(struct fs_file_t *zfp, const char *file_name,
                      fs_mode_t mode)
{
        FRESULT res;
        uint8_t fs_mode;
        void *ptr;

        if (k_mem_slab_alloc(&fatfs_filep_pool, &ptr, K_NO_WAIT) == 0) {
                (void)memset(ptr, 0, sizeof(FIL));
                zfp->filep = ptr;
        } else {
                return -ENOMEM;
        }

        fs_mode = translate_flags(mode);

        res = f_open(zfp->filep, translate_path(file_name), fs_mode);

        if (res != FR_OK) {
                k_mem_slab_free(&fatfs_filep_pool, ptr);
                zfp->filep = NULL;
        }

        return translate_error(res);
}
```

FAT Filesystem Example

subsys/fs/fat_fs.c

```
static int fatfs_init(void)
{

        return fs_register(FS_FATFS, &fatfs_fs);
}

SYS_INIT(fatfs_init, POST_KERNEL, CONFIG_FILE_SYSTEM_INIT_PRIORITY);
```

FAT Filesystem Example

Kconfig driven (not devicetree)

```
menu "File Systems"

config FILE_SYSTEM
        bool "File system support"
        help
          Enables support for file system.

if FILE_SYSTEM

module = FS
module-str = fs
subsys/fs/Kconfig
        depends on ARCH_POSIX
        help
          Expose file system partitions to the host system through FUSE.

rsource "Kconfig.fatfs"
rsource "Kconfig.littlefs"
rsource "ext2/Kconfig"

endif # FILE_SYSTEM
```

FAT Filesystem Example

- Kconfig driven (not devicetree)

```
menu "File Systems"

config FILE_SYSTEM
        bool "File system support"
        help
          Enables support for file system.

if FILE_SYSTEM

module = FS
module-str = fs
subsys/fs/Kconfig
        depends on ARCH_POSIX
        help
          Expose file system partitions to the host system through FUSE.

rsource "Kconfig.fatfs"
rsource "Kconfig.littlefs"
rsource "ext2/Kconfig"

endif # FILE_SYSTEM
```

SD Card Example

```
static const struct sdhc_driver_api sdhc_spi_api = {
        .request = sdhc_spi_request,
        .set_io = sdhc_spi_set_io,
        .get_host_props = sdhc_spi_get_host_props,
        .get_card_present = sdhc_spi_get_card_present,
        .reset = sdhc_spi_reset,
        .card_busy = sdhc_spi_card_busy,
};
```

drivers/sdhc/sdhc_spi.c

SD Card Example

- Macro fun 🫠

```
#define SDHC_SPI_INIT(n)                                                    \
        const struct sdhc_spi_config sdhc_spi_config_##n = {                \
                .spi_dev = DEVICE_DT_GET(DT_INST_PARENT(n)),                \
                .pwr_gpio = GPIO_DT_SPEC_INST_GET_OR(n, pwr_gpios, {0}),    \
                .spi_max_freq = DT_INST_PROP(n, spi_max_frequency),         \
                .power_delay_ms = DT_INST_PROP(n, power_delay_ms),          \
        };                                                                  \
                                                                            \
        struct sdhc_spi_data sdhc_spi_data_##n = {                          \
                .cfg_a = SPI_CONFIG_DT_INST(n,                              \
                                (SPI_LOCK_ON | SPI_HOLD_ON_CS | SPI_WORD_SET(8) \
                                 | (DT_INST_PROP(n, spi_clock_mode_cpol) ? SPI_MODE_CPOL : 0) \
                                 | (DT_INST_PROP(n, spi_clock_mode_cpha) ? SPI_MODE_CPHA : 0) \
                                ),\
                                0),                                         \
        };                                                                  \
                                                                            \
        DEVICE_DT_INST_DEFINE(n,                                            \
                        &sdhc_spi_init,                                     \
                        NULL,                                               \
                        &sdhc_spi_data_##n,                                 \
                        &sdhc_spi_config_##n,                               \
                        POST_KERNEL,                                        \
                        CONFIG_SDHC_INIT_PRIORITY,                          \
                        &sdhc_spi_api);

DT_INST_FOREACH_STATUS_OKAY(SDHC_SPI_INIT)
```

SD Card Example

- Driven by devicetree (not really Kconfig)

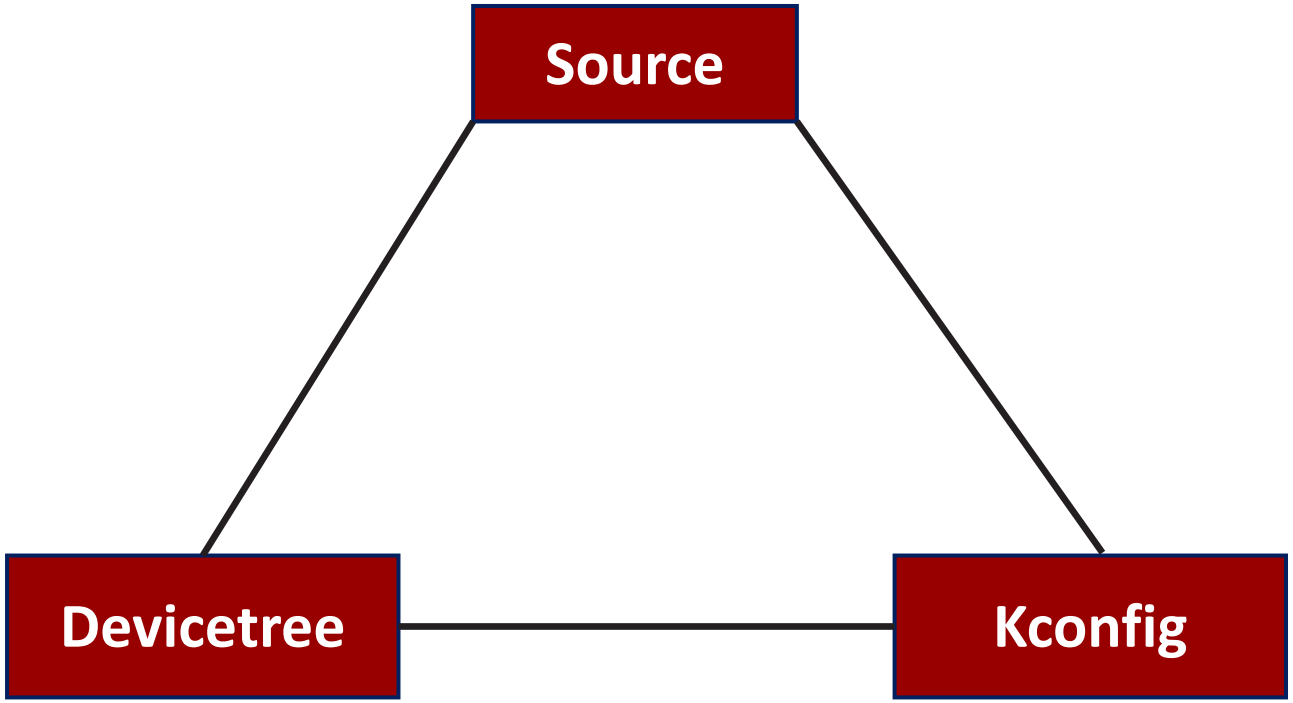
```
# Copyright (c) 2022, NXP
# SPDX -License-Identifier: Apache-2.0

config SPI_SDHC
        bool "SD protocol over SPI bus"
        default y
        depends on DT_HAS_ZEPHYR_SDHC_SPI_SLOT_ENABLED
        select SPI
        select CRC
        select SDHC_SUPPORTS_SPI_MODE
        help
          Enable the SPI SD host controller driver
```

Walkthrough Using Out-of-Tree Driver

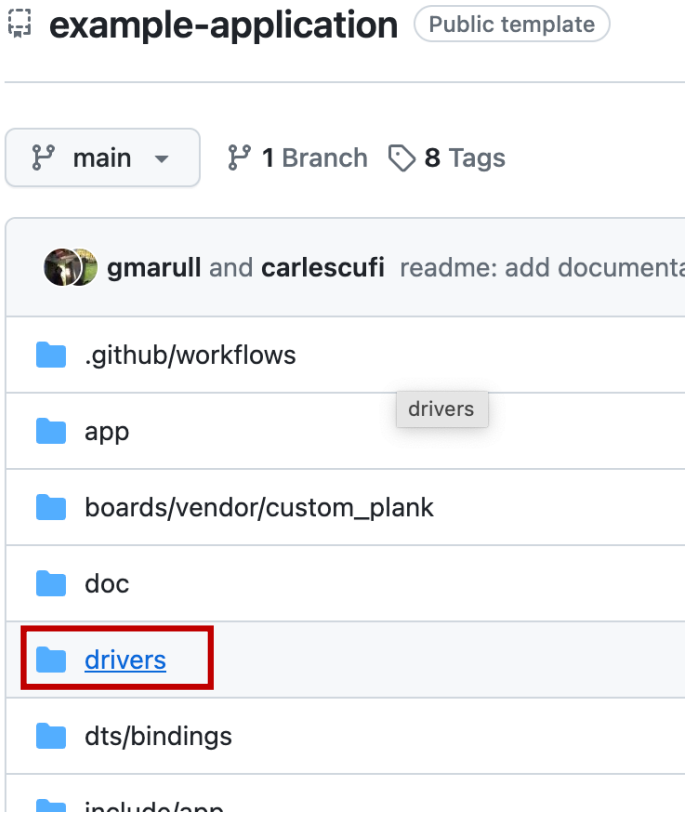
- Why?
- Out-of-tree drivers are best:
  - To avoid forking upstream
    - Messy
    - Have to stay up-to-date
  - To avoid having to wait for your driver to be mainlined

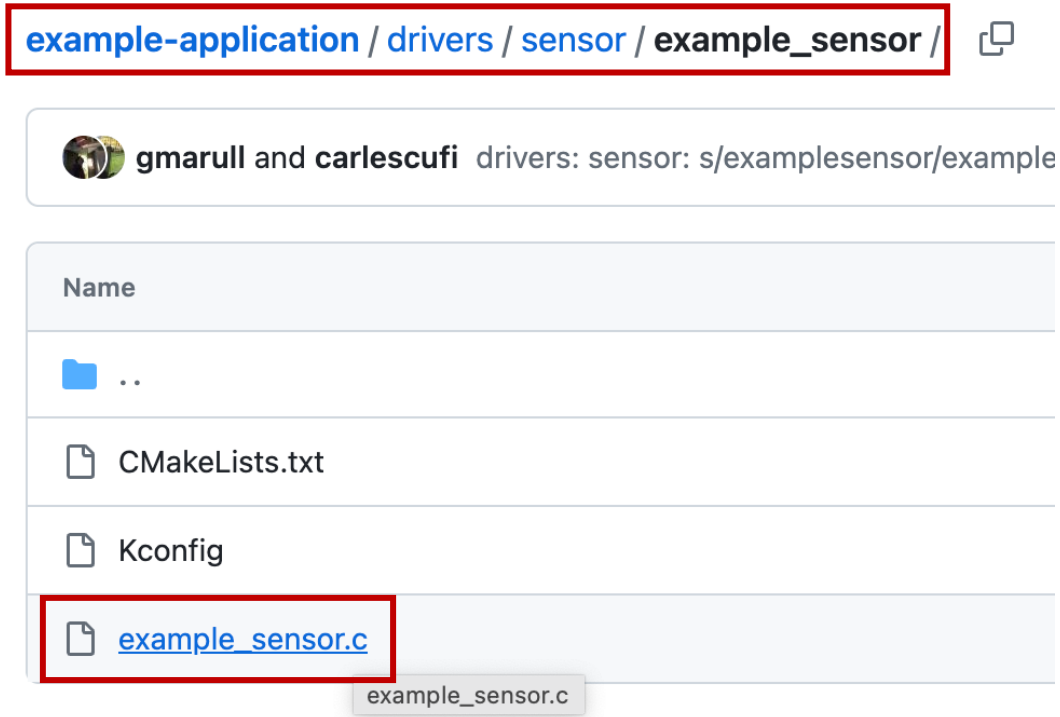
https://github.com/zephyrproject-rtos/example-application

Zephyr Triumverate



Source Code (Caesar)!





Kconfig (Pompey)!

MAB LABS
Embedded Solutions

example-application / Kconfig

Gregory Shue and carlescufi lib: create empty lib subsystem

Code Blame 9 lines (8 loc) · 349 Bytes

```
1  # Copyright (c) 2021 Nordic Semiconductor ASA
2  # SPDX-License-Identifier: Apache-2.0
3  #
4  # This Kconfig file is picked by the Zephyr build system because it is defined
5  # as the module Kconfig entry point (see zephyr/module.yml). You can browse
6  # module options by going to Zephyr -> Modules in Kconfig.
7
8  rsource "drivers/Kconfig"
9  rsource "lib/Kconfig"
```

example-application / drivers / Kconfig

gmarull and carlescufi drivers: blink: add custom out-of-tree driver class

Code Blame 7 lines (6 loc) · 157 Bytes

```
1  # Copyright (c) 2021 Nordic Semiconductor ASA
2  # SPDX-License-Identifier: Apache-2.0
3
4  menu "Drivers"
5  rsource "blink/Kconfig"
6  rsource "sensor/Kconfig"
7  endmenu
```

example-application / drivers / sensor / Kconfig

gmarull and carlescufi drivers: sensor: s/examplesensor/example-|_sensor/

Code Blame 6 lines (5 loc) · 143 Bytes

```
1  # Copyright (c) 2021 Nordic Semiconductor ASA
2  # SPDX-License-Identifier: Apache-2.0
3
4  if SENSOR
5  rsource "example_sensor/Kconfig"
6  endif # SENSOR
```

Kconfig (Pompey)!

MAB LABS
Embedded Solutions

example-application / drivers / sensor / example_sensor / **Kconfig**

gmarull and **carlescufi** drivers: sensor: s/examplesensor/example-

Code | Blame | 10 lines (9 loc) · 234 Bytes

```
1   # Copyright (c) 2021 Nordic Semiconductor ASA
2   # SPDX-License-Identifier: Apache-2.0
3
4   config EXAMPLE_SENSOR
5           bool "Example sensor"
6           default y
7           depends on DT_HAS_ZEPHYR_EXAMPLE_SENSOR_ENABLE
8           select GPIO
9           help
10            Enable example sensor
```

Devicetree (Crassus)!

MAB LABS
Embedded Solutions

example-application / dts / bindings / sensor / zephyr,example-sensor.yaml

gmarull and carlescufi drivers: sensor: s/examplesensor/example-|_sensor/

Code Blame 23 lines (17 loc) · 601 Bytes

```
1   # Copyright (c) 2021 Nordic Semiconductor ASA
2   # SPDX-License-Identifier: Apache-2.0
3
4   description: |
5     An example sensor that reads the GPIO level defined in input-gpios. The
6     purpose of this sensor is to demonstrate how to create out-of-tree drivers.
7
8     Example definition in devicetree:
9
10      example-sensor {
11          compatible = "zephyr,example-sensor";
12          input-gpios = <&gpio0 0 (GPIO_PULL_UP | GPIO_ACTIVE_LOW)>;
13      };
14
15  compatible: "zephyr,example-sensor"
16
17  include: base.yaml
18
19  properties:
20    input-gpios:
21      type: phandle-array
22      required: true
23      description: Input GPIO to be sensed.
```

Summary

- Learned how the Zephyr device driver model enables easy reuse
- Learned how it allows us to leverage external repositories to handle most of driver logic
  - Simply need a thin wrapper to fit within Zephyr's subsystem API
- Zephyr Triumverate: Source code, Kconfig, Devicetree
  - Saw how different drivers leverage these constructs
- Walked through an out-of-tree driver as an example

MAB LABS

Embedded Solutions

Thank you!

Questions?

MOHAMMED BILLOO

Embedded World 202