Zephyr device model

Each instance of a driver is represented in a **struct device**.

```
struct device_config
{
    const char *name;
    int (*init) (struct device *device);
    const void* config_info;
};

struct device
{
    struct device_config *config;
    const void *driver_api;
    void *driver_data;
}
```

The above code has

- An initialization function
- It's name
- Exported API
- Dedicated data and configurations

Defining a new device is equivalent to adding a device in a device section (array of sorted devices). This is done using just one macro.

```
DEVICE_AND_API_INIT(
    dev_name, drv_name,
    init_fn, data, cfg_info,
    level, prio, api);
```

From a bird's view, we can say that **Zephyr application uses APIs to ask the device drivers to perform tasks on actual hardware.**

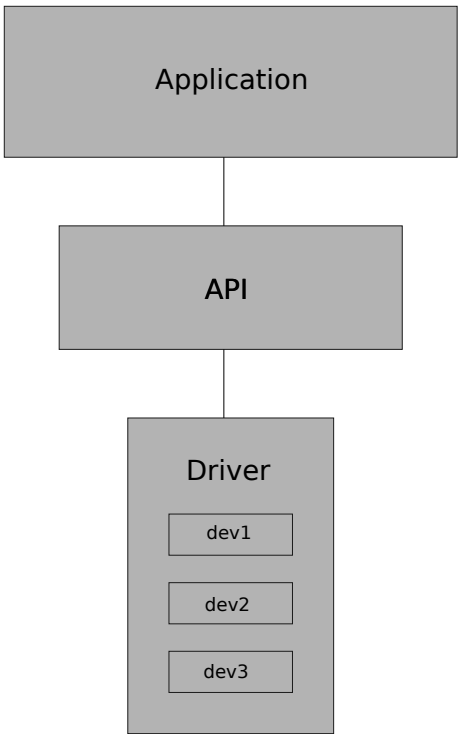

Figure 16: High-level Driver Model

Since Zephyr supports many devices, APIs, and drivers, there is a **.conf** file which specifies what drivers are required, to the build system.

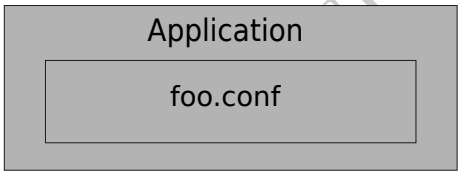

Figure 17: Configuration settings

To build applications for specific boards, devices must be allocated inside the driver source files which is taken care of by **devicetrees**. During development, devicetree overlay files are used.

The development process can be described in the following manner:

- A base devicetree, which is a tree data structure is provided by the board (vendor).
- Zephyr is build for that board which has a devicetree as mentioned above.
- Devicetree overlay file is used to modify the devicetree for that particular application.

- The device driver source code consume the final modified devicetree and then allocates the **struct devices**.
- Also, the applications use the same devicetree to access the devices.

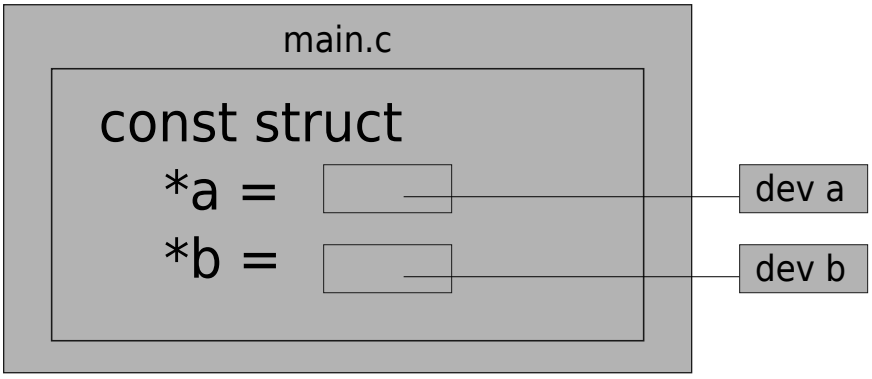

Figure 18: Getting device pointers

The following section provides a brief explanation to some of the device drivers.