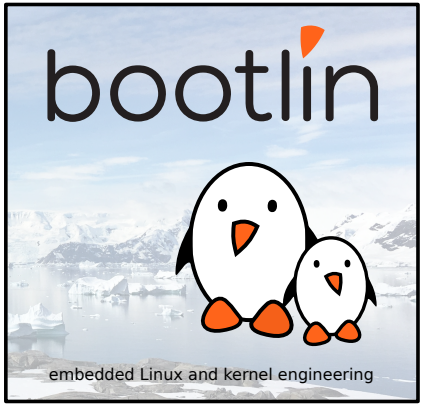
An Intro to Zephyr Driver Development

Miguel Gazquez
miguel.gazquez@bootlin.com



© Copyright 2004-2025, Bootlin.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!

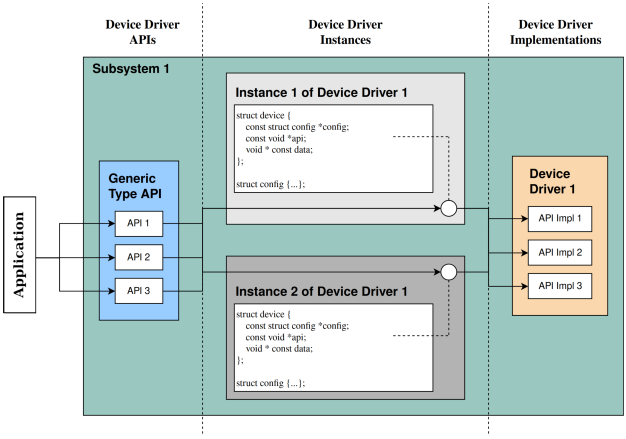
Miguel Gazquez

- Embedded Zephyr and Linux engineer at Bootlin
  - Zephyr **expertise**
    - Implemented and contributed upstream multiple drivers: a sensor, a display controller...
    - Support for the RISC-V CH32v303 SoC
  - **Development**, consulting and training
  - Strong open-source focus
- Open-source contributor
- Living in **Toulouse**, France

Zephyr

- Real-Time OS
- Lightweight & Scalable
  - "Hello World" needs ≈16Ko Flash and 4Ko RAM
- Feature-rich
- Wide Ecosystem Support
  - More than 700 supported boards
- Open source and community driven

Device Driver Model



source: https://docs.zephyrproject.org/latest/kernel/drivers/index.html

Using drivers from an application

```c
int main(void)
{
  struct sensor_value accel_x, accel_y, accel_z;
  int ret;

  const struct device *dev = DEVICE_DT_GET(DT_ALIAS(accel0));

  while (1) {
    sensor_sample_fetch(dev);

    sensor_channel_get(dev, SENSOR_CHAN_ACCEL_X, &accel_x);
    sensor_channel_get(dev, SENSOR_CHAN_ACCEL_Y, &accel_y);
    sensor_channel_get(dev, SENSOR_CHAN_ACCEL_Z, &accel_z);

    printk("%16s [m/s^2]:    (%12.6f, %12.6f, %12.6f)\n", dev->name,
           sensor_value_to_double(&accel_x), sensor_value_to_double(&accel_y),
           sensor_value_to_double(&accel_z));

    k_msleep(1000);
  }

  return 0;
}
```

Example 1 : A driver for the Nunchuk

Adding your device to the DeviceTree

app.overlay

```
&i2c1 {
    nunchuk: nunchuk@52 {
        reg = <0x52>;
        compatible = "nintendo,nunchuk";
        polling_interval_ms = <50>;
    };
};
```

Adding your driver to the build system

drivers/input/Kconfig.nunchuk

```
config INPUT_NUNCHUK
  bool "Nintendo Nunchuk joystick"
  default y
  depends on DT_HAS_NINTENDO_NUNCHUK_ENABLED
  select I2C
  help
    This option enables the driver for the Nintendo Nunchuk joystick.
```

drivers/input/CMakeLists.txt

```
...
zephyr_library_sources_ifdef(CONFIG_INPUT_NUNCHUK input_nunchuk.c)
...
```

Create the driver file

drivers/input/input_nunchuk.c

```
struct nunchuk_config {
    struct i2c_dt_spec i2c_bus;
    int polling_interval_ms;
};

struct nunchuk_data {
    uint8_t joystick_x;
    uint8_t joystick_y;
    bool button_c;
    bool button_z;
    [...]
};

static int nunchuk_init(const struct device *dev)
{
    [...]
}
```

Create the struct device

drivers/input/input_nunchuk.c

```
#define DT_DRV_COMPAT nintendo_nunchuk

// Code of the driver

#define NUNCHUK_INIT(inst)                                                  \
    static const struct nunchuk_config nunchuk_config_##inst = {           \
        .i2c_bus = I2C_DT_SPEC_INST_GET(inst),                              \
        .polling_interval_ms = DT_INST_PROP(inst, polling_interval_ms),     \
    };                                                                      \
                                                                            \
    static struct nunchuk_data nunchuk_data_##inst;                         \
                                                                            \
    DEVICE_DT_INST_DEFINE(inst, &nunchuk_init, NULL, &nunchuk_data_##inst,  \
              &nunchuk_config_##inst, POST_KERNEL, CONFIG_INPUT_INIT_PRIORITY,  \
              NULL);

DT_INST_FOREACH_STATUS_OKAY(NUNCHUK_INIT)
```

The init function

```
static int nunchuk_init(const struct device *dev)
{
  const struct nunchuk_config *cfg = dev->config;
  struct nunchuk_data *data = dev->data;

  uint8_t init_seq_1[2] = {0xf0, 0x55};
  uint8_t init_seq_2[2] = {0xfb, 0x00};
  uint8_t buffer[NUNCHUK_READ_SIZE];

  if (!i2c_is_ready_dt(&cfg->i2c_bus)) {
    return -ENODEV;
  }

  i2c_write_dt(&cfg->i2c_bus, init_seq_1, sizeof(init_seq_1));
  k_msleep(1);
  i2c_write_dt(&cfg->i2c_bus, init_seq_2, sizeof(init_seq_2));

  [...]
  k_work_init_delayable(&data->work, nunchuk_poll);
  return k_work_reschedule(&data->work, data->interval_ms);
}
```

The poll function

```c
static void nunchuk_poll(struct k_work *work)
{
  const struct nunchuk_config *cfg = dev->config;
  struct k_work_delayable *dwork = k_work_delayable_from_work(work);
  struct nunchuk_data *data = CONTAINER_OF(dwork, struct nunchuk_data, work);
  const struct device *dev = data->dev;
  uint8_t buffer[NUNCHUK_READ_SIZE];
  uint8_t joystick_x;

  i2c_write_dt(&cfg->i2c_bus, &value, sizeof(value));
  k_msleep(NUNCHUK_DELAY_MS);
  i2c_read_dt(&cfg->i2c_bus, buffer, NUNCHUK_READ_SIZE);

  joystick_x = buffer[0];

  if (joystick_x != data->joystick_x) {
    data->joystick_x = joystick_x;
    input_report_abs(dev, INPUT_ABS_X, data->joystick_x, true, K_FOREVER);
  }
  [...]
  k_work_reschedule(dwork, data->interval_ms);
}
```

Example 2 : The lsm9ds1 sensor

Leveraging the HAL

Drivers can use Hardware Abstraction Layers

- Provided by the silicon vendors.
- Enables faster driver development
- Used in other projects: more maturity

But

- Potential API incompatibilities
- Maintained elsewhere
- Variability between vendors

Using the Hal

drivers/sensor/st/lsm9ds1/lsm9ds1.c

```
static int lsm9ds1_sample_fetch(const struct device *dev, enum sensor_channel chan)
{
  switch (chan) {
  case SENSOR_CHAN_ACCEL_XYZ:
    lsm9ds1_sample_fetch_accel(dev);
    break;
  [...]
  }
}

static int lsm9ds1_sample_fetch_accel(const struct device *dev)
{
  const struct lsm9ds1_config *cfg = dev->config;
  stmdev_ctx_t *ctx = (stmdev_ctx_t *)&cfg->ctx;
  struct lsm9ds1_data *data = dev->data;
  int ret;

  ret = lsm9ds1_acceleration_raw_get(ctx, data->acc);

  [...]
}
```

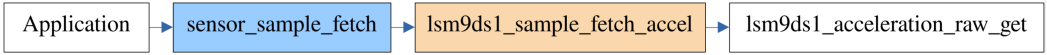
Defining the struct api

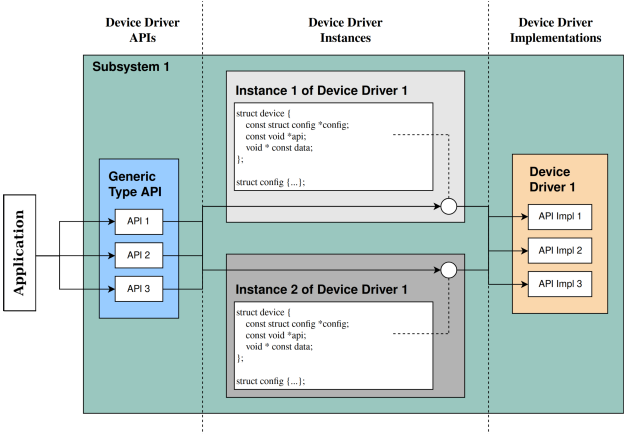
drivers/sensor/st/lsm9ds1/lsm9ds1.c

```
static DEVICE_API(sensor, lsm9ds1_api_funcs) = {
  .sample_fetch = lsm9ds1_sample_fetch,
  .channel_get = lsm9ds1_channel_get,
  .attr_set = lsm9ds1_attr_set,
};

#define LSM9DS1_DEFINE(inst)                                                    \
                                                                                \
  // Declaration of the structs config and data                                 \
                                                                                \
  SENSOR_DEVICE_DT_INST_DEFINE(inst, lsm9ds1_init, NULL, &lsm9ds1_data_##inst,  \
            &lsm9ds1_config_##inst, POST_KERNEL, CONFIG_SENSOR_INIT_PRIORITY,  \
            &lsm9ds1_api_funcs);
```

Bringing It Together: Driver & Device Model





Additional Reading

- Code shown in this talk
  - `drivers/input/input_nunchuk.c`
  - `drivers/sensor/st/lsm9ds1/lsm9ds1.c`
- Blog posts I wrote
  - Getting started with Zephyr
  - Understanding Zephyr's Blinky Sample
  - Implementing a device driver for a sensor (lsm9ds1)
  - Integrating ST7789H2 Display Support on STM32L562E-DK with Zephyr: A Step-by-Step Guide
  - *More to come...*
- The official documentation

Questions? Suggestions? Comments?

Miguel Gazquez
miguel.gazquez@bootlin.com

Slides under CC-BY-SA 3.0
https://bootlin.com/pub/conferences/2025/meetup/gazquez-driver-development-for-zephyr/