Zephyr® Project
Developer Summit

Streamline Testing with Emulator Backend APIs and Generic Tests

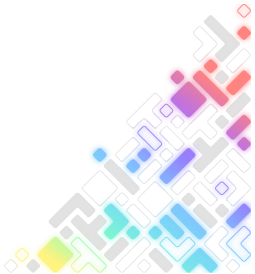
Tristan Honscheid
Google

THE LINUX FOUNDATION

#EmbeddedOSSummit

Overview

1. Background on Zephyr Emulators
2. Generic Tests and the Backend API
3. Implementation example



About Me

- Firmware engineer at Google in ChromeOS
- Collaborator in Zephyr Sensors, have also contributed to Twister / testing.

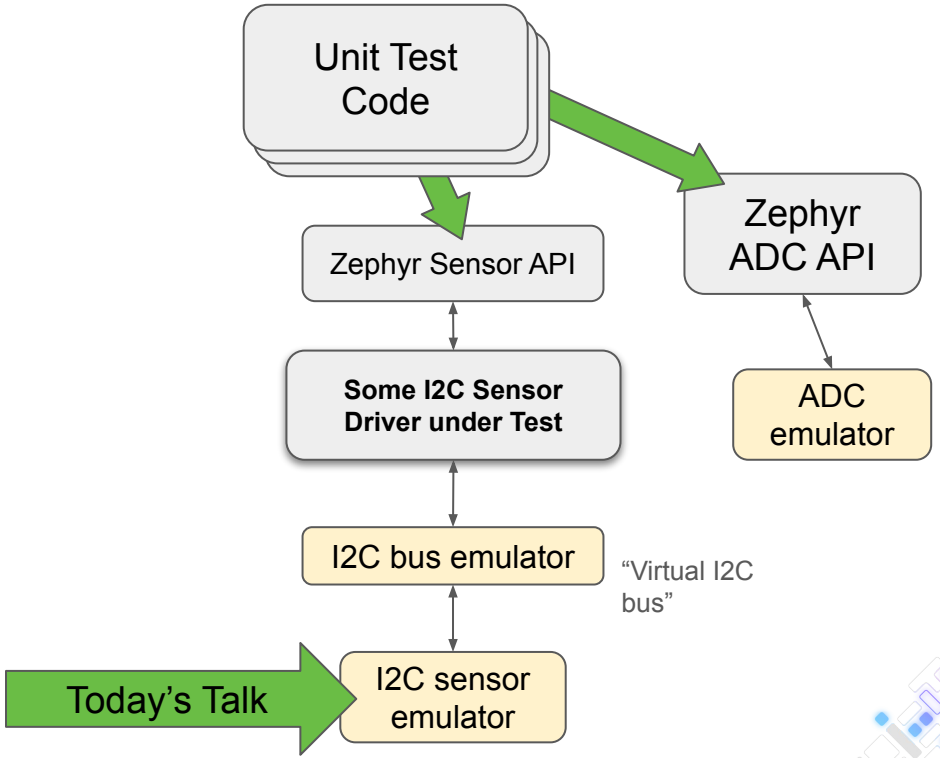
- We use Zephyr in the ChromeOS Embedded Controller
  - System management (USB-PD, charging, thermal, keyboard, etc)
  - Open Source!

chromeOS

Zephyr’s Testing and Emulation Framework

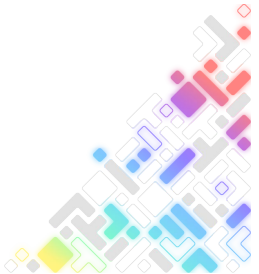
Unit Test Firmware on a PC with Emulators

- Emulators are mock hardware devices
  - Many sensors, but also GPIO, I2C, ADC, etc.
  - Coded in C, compiled in with tests
  - Typically native_posix/native_sim
  - Assembled together via Devicetree
- Validate your driver and even your application code without hardware (great for CI)

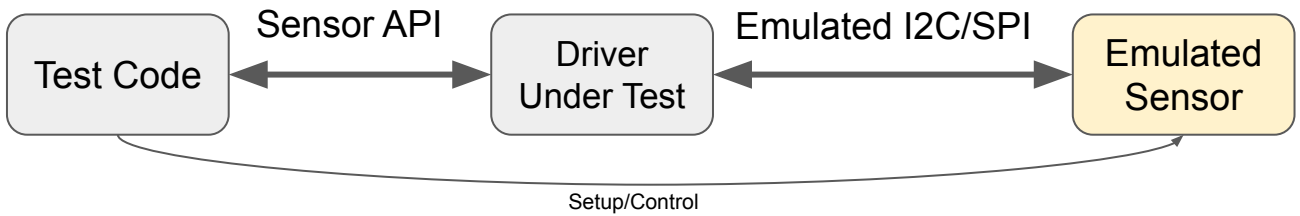


Testing and Emulation is Good

- Zephyr users should have confidence drivers work as intended
- Unfortunately, bugs do occur
  - Common issues: edge cases / range, unutilized features
- Unit testing against emulators is best strategy for call code paths in a driver
  - How can we make this easier?



Basic Test + Emulator Flow



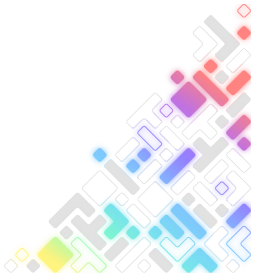
1. Test code uses some mechanism to configure registers on the emulator:
   `emul_mytempsensor_set_reg(0x45, 0x1A87);`
2. Use sensor API to take a sample (fetch/get)
3. Compare against the expected value

This is the only step unique to a device.

Sensor Emul Backend API

Sensor Emulator Backend API

- Standardize the mechanism for setting an expected value on the emulator
  - Use real, SI units
  - Emulator must handle converting SI to register coding
- Discoverability of supported channels and ranges
- Enable a “generic test” that can test a sensor without knowing anything about it
  - Concept can be expanded to other types of peripheral tests



Sensor Emulator Backend API

include/zephyr/drivers/emul_sensor.h

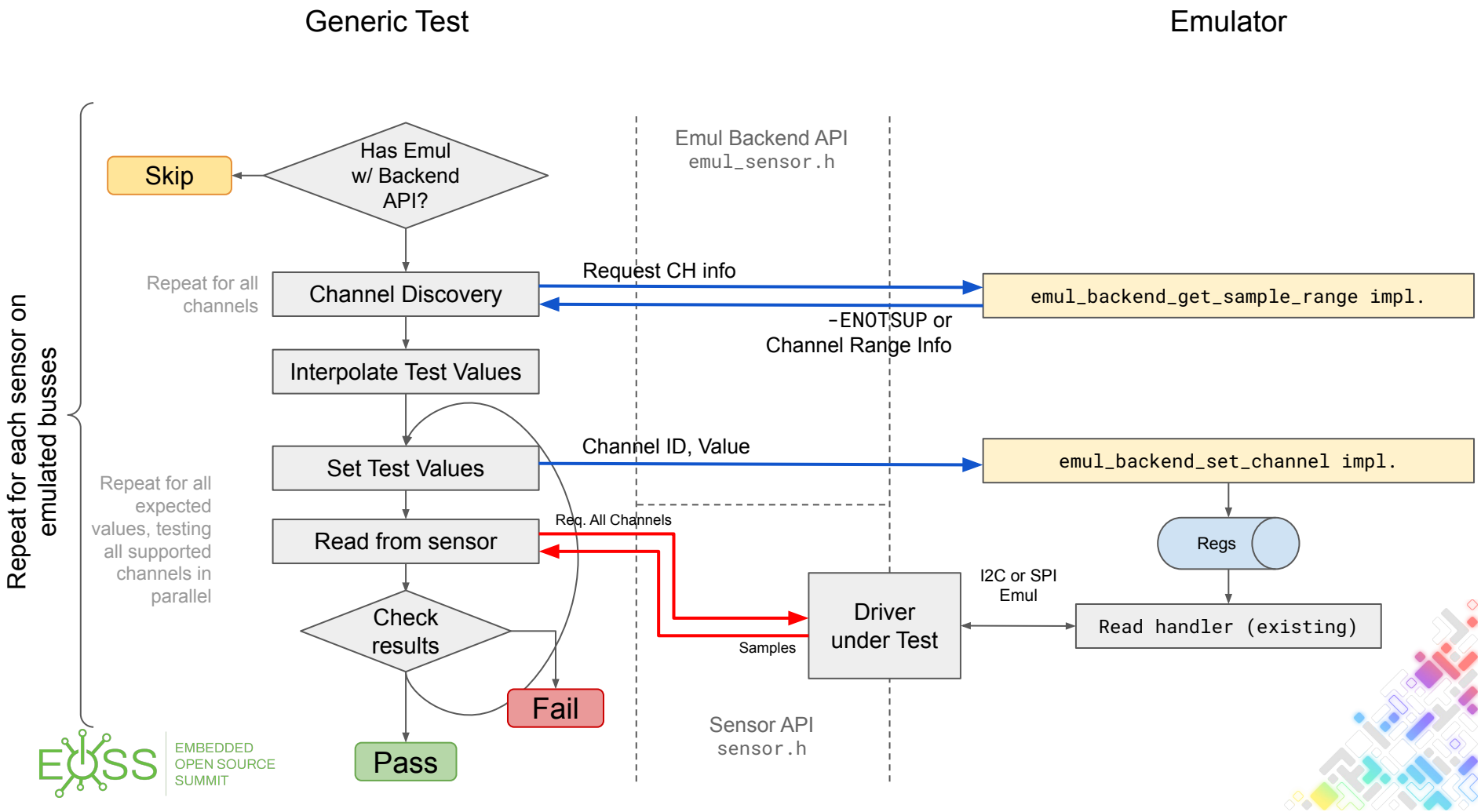
```
__subsystem struct emul_sensor_backend_api {
	/** Sets a given fractional value for a given sensor channel. */
	int (*set_channel)(const struct emul *target, enum sensor_channel ch, const q31_t *value,
			   int8_t shift);
	/** Retrieve a range of sensor values to use with test. */
	int (*get_sample_range)(const struct emul *target, enum sensor_channel ch, q31_t *lower,
				q31_t *upper, q31_t *epsilon, int8_t *shift);









};
```

Sensor Emulator Backend API

include/zephyr/drivers/emul_sensor.h

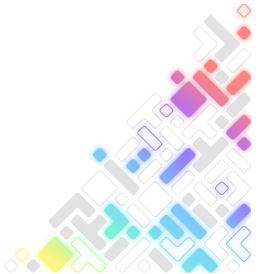
```
__subsystem struct emul_sensor_backend_api {
	/** Sets a given fractional value for a given sensor channel. */
	int (*set_channel)(const struct emul *target, enum sensor_channel ch, const q31_t *value,
			   int8_t shift);
	/** Retrieve a range of sensor values to use with test. */
	int (*get_sample_range)(const struct emul *target, enum sensor_channel ch, q31_t *lower,
				q31_t *upper, q31_t *epsilon, int8_t *shift);


	/** Set the attribute value(s) of a given chanel. */
	int (*set_attribute)(const struct emul *target, enum sensor_channel ch,
			     enum sensor_attribute attribute, const void *value);
	/** Get metadata about an attribute. */
	int (*get_attribute_metadata)(const struct emul *target, enum sensor_channel ch,
				      enum sensor_attribute attribute, q31_t *min, q31_t *max,
				      q31_t *increment, int8_t *shift);
};
```

Generic Test


Generic Test
Emulator
Repeat for each sensor on emulated busses
Skip
Has Emul w/ Backend API?
Emul Backend API
emul_sensor.h
Repeat for all channels
Channel Discovery
Request CH info
emul_backend_get_sample_range impl.
-ENOTSUP or Channel Range Info
Interpolate Test Values
Repeat for all expected values, testing all supported channels in parallel
Set Test Values
Channel ID, Value
emul_backend_set_channel impl.
Read from sensor
Req. All Channels
Regs
I2C or SPI Emul
Driver under Test
Samples
Read handler (existing)
Check results
Fail
Pass
Sensor API
sensor.h
EMBEDDED OPEN SOURCE SUMMIT

Aside: Fixed point numbers and Sensors V2

- What is `q31_t` and `int8_t shift`?
- The Generic Sensor Test and Sensor Emul Backend API are built around the newer Zephyr Sensors V2 API
  - No longer uses the two-part `struct sensor_value` object
  - Use 32-bit fixed point numbers with SI units
  - Benefits: easier arithmetic and processing, higher throughput using async RTIO tech, compatible with DSP subsystem. See link at end for additional info.



Aside: Fixed point numbers and Sensors V2

$-2^3 = -8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	$2^{-1} = 1/2$	$2^{-2} = 1/4$	$2^{-3} = 1/8$	$2^{-4} = 1/16$
1	0	0	1	0	1	0	0

= -6.75

- Fixed point math is very common in the DSP space
- Bits right of the arbitrary decimal point have fractional weights
- Decimal point location is defined by the `shift`. Set `shift` based on the sample range. (Slightly different from Q*n.m* notation)
- Sensors V2 uses 32-bit signed containers

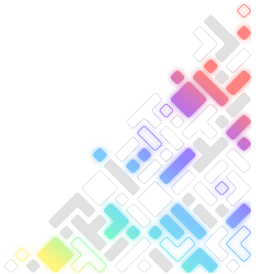
Aside: Fixed point numbers and Sensors V2

Shift value (Sensors V2 API)	Min Value = -2^{shift}	Max Value = (2^{shift} - $2^{-31+shift}$)
-2	-0.25	0.24999999988358468
-1	-0.5	0.49999999976716936
0	-1	0.9999999995343387
1	-2	1.9999999990686774
2	-4	3.999999998137355
31	-2,147,483,648	-2,147,483,647

Examples of shifts and ranges

Aside: Fixed point numbers and Sensors V2

- Suppose you have a temperature sensor with range **`-10℃`** to **`100℃`**
- Which shift to use?
  - Shift of 7 → [-128, 127.99999994039536]
- Temperature of **`50℃`** converts to:
  - $(50 * 2^{31}) >> 7 = 838860800 = 0x32000000$
  - $(838860800 << 7) / 2^{31} = 50.0$
  - We'll check out some C code soon.
- **Watch out for overflow** – order operations carefully and maybe use `int64_t`



Tour of an implementation

(AKM09918C digital compass)

Adding backend API impl. to the AKM09928C (1/4)

- Get sample range returns info about each supported channel, or `-ENOTSUP`
- Epsilon = allowed error (due to sensor accuracy limits)

```
static int akm09918c_emul_backend_get_sample_range(const struct emul *target,
                enum sensor_channel ch, q31_t *lower,
                q31_t *upper, q31_t *epsilon, int8_t *shift)
{
  ARG_UNUSED(target);

  if (!lower || !upper || !epsilon || !shift) {
    return -EINVAL;
  }

  switch (ch) {
  case SENSOR_CHAN_MAGN_X:
  case SENSOR_CHAN_MAGN_Y:
  case SENSOR_CHAN_MAGN_Z:
    /* +/- 49.12 Gs is the measurement range. 0.0015 Gs is the granularity */
    *shift = 6;
    *upper = (int64_t)(49.12 * ((int64_t)INT32_MAX + 1)) >> *shift;
    *lower = -*upper;
    *epsilon = (int64_t)(0.0015 * ((int64_t)INT32_MAX + 1)) >> *shift;
    break;
  default:
    return -ENOTSUP;
  }

  return 0;
}
```

Adding backend API impl. to the AKM09928C (2/4)

- Check inputs
- Determine appropriate internal registers to update
- Individual channels, no XYZ tuples.

```
static int akm09918c_emul_backend_set_channel(const struct emul *target, enum sensor_channel ch,
                const q31_t *value, int8_t shift)
{
  if (!target || !target->data) {
    return -EINVAL;
  }

  struct akm09918c_emul_data *data = target->data;
  uint8_t reg;

  switch (ch) {
  case SENSOR_CHAN_MAGN_X:
    reg = AKM09918C_REG_HXL;
    break;
  case SENSOR_CHAN_MAGN_Y:
    reg = AKM09918C_REG_HYL;
    break;
  case SENSOR_CHAN_MAGN_Z:
    reg = AKM09918C_REG_HZL;
    break;
  /* This function only supports setting single channels, so skip MAGN_XYZ */
  default:
    return -ENOTSUP;
  }
```

Adding backend API impl. to the AKM09928C (3/4)

- Convert SI units to reg. Values
- Remember to update any “data ready” bits
- Clamp to allowed values

```
  /* Set the ST1 register to show we have data */
  data->reg[AKM09918C_REG_ST1] |= AKM09918C_ST1_DRDY;

  /* Convert fixed-point Gauss values into microgauss and then into its bit representation */
  int32_t microgauss =
    (shift < 0 ? ((int64_t)*value >> -shift) : ((int64_t)*value << shift)) * 1000000 /
    ((int64_t)INT32_MAX + 1);

  int16_t reg_val =
    CLAMP(microgauss, AKM09918C_MAGN_MIN_MICRO_GAUSS, AKM09918C_MAGN_MAX_MICRO_GAUSS) /
    AKM09918C_MICRO_GAUSS_PER_BIT;

  /* Insert reading into registers */
  data->reg[reg] = reg_val & 0xFF;
  data->reg[reg + 1] = (reg_val >> 8) & 0xFF;

  return 0;
}
```

Adding backend API impl. to the AKM09928C (4/4)

```
static const struct emul_sensor_backend_api akm09918c_emul_sensor_backend_api = {
  .set_channel = akm09918c_emul_backend_set_channel,
  .get_sample_range = akm09918c_emul_backend_get_sample_range,
};

#define AKM09918C_EMUL(n)
  const struct akm09918c_emul_cfg akm09918c_emul_cfg_##n;
  struct akm09918c_emul_data akm09918c_emul_data_##n;
  EMUL_DT_INST_DEFINE(n, akm09918c_emul_init, &akm09918c_emul_data_##n,
          &akm09918c_emul_cfg_##n, &akm09918c_emul_api_i2c,                \
          &akm09918c_emul_sensor_backend_api)
```

- Declare struct w/ func popinters
- Register it in `EMUL_DT_INST_DEFINE()`

Generic Test

- Lives under the Sensor “build_all” test
  - tests/drivers/build_all/sensor/src/generic_test.c
- This test has a devicetree with nodes for all sensors (i2c.dtsi, spi.dtsi, etc)

```
#define DECLARE_ZTEST_PER_DEVICE(n)
  ZTEST(generic, test_##n)
  {
    run_generic_test(DEVICE_DT_GET(n));
  }

/* Iterate through each of the emulated buses and create a test for each device. */
DT_FOREACH_CHILD_STATUS_OKAY(DT_NODELABEL(test_i2c), DECLARE_ZTEST_PER_DEVICE)
DT_FOREACH_CHILD_STATUS_OKAY(DT_NODELABEL(test_i3c), DECLARE_ZTEST_PER_DEVICE)
DT_FOREACH_CHILD_STATUS_OKAY(DT_NODELABEL(test_spi), DECLARE_ZTEST_PER_DEVICE)

ZTEST_SUITE(generic, NULL, NULL, before, NULL, NULL);
```

Generic Test - Discover emulator support

- Skip drivers if no emulator or no backend API impl. exists
- All drivers will initialize as Zephyr boots

```c
/**
 * @brief Helper function the carries out the generic sensor test for a given sensor device.
 *        Verifies that the device has a suitable emulator that implements the backend API and
 *        skips the test gracefully if not.
 */
static void run_generic_test(const struct device *dev)
{
  zassert_not_null(dev, "Cannot get device pointer. Is this driver properly instantiated?");

  const struct emul *emul = emul_get_binding(dev->name);

  /* Skip this sensor if there is no emulator loaded. */
  if (!emul) {
    ztest_test_skip();
  }

  /* Also skip if this emulator does not implement the backend API. */
  if (!emul_sensor_backend_is_supported(emul)) {
    ztest_test_skip();
  }
```

Generic Test - Channel discovery

- Table of channel information
- Iterate through all known channels, record info about which are supported
- Generate linearly-interpolated test points throughout range

Note: some lines dropped for brevity

```
/* Discover supported channels on this device and fill out our sensor read request */
for (enum sensor_channel ch = 0; ch < ARRAY_SIZE(channel_table); ch++) {
  if (SENSOR_CHANNEL_3_AXIS(ch)) {
    continue;
  }

  q31_t lower, upper;
  int8_t shift;

  if (emul_sensor_backend_get_sample_range(emul, ch, &lower, &upper,
              &channel_table[ch].epsilon, &shift) == 0) {
    /* This channel is supported */
    channel_table[ch].supported = true;

    /* Generate a set of CONFIG_GENERIC_SENSOR_TEST_NUM_EXPECTED_VALS test
     * values.
     */

    channel_table[ch].expected_value_shift = shift;
    for (size_t i = 0; i < CONFIG_GENERIC_SENSOR_TEST_NUM_EXPECTED_VALS; i++) {
      channel_table[ch].expected_values[i] =
        lower +
        (i * ((int64_t)upper - lower) /
         (CONFIG_GENERIC_SENSOR_TEST_NUM_EXPECTED_VALS - 1));
```

Generic Test - Set expected values

- Use backend API to configure emulator
- One set of expected values across supported channels at a time

```
for (size_t iteration = 0; iteration < CONFIG_GENERIC_SENSOR_TEST_NUM_EXPECTED_VALS;
     iteration++) {
  int rv;

  /* Set this iteration's expected values in emul for every supported channel */
  for (size_t i = 0; i < iodev_read_config.count; i++) {
    enum sensor_channel ch = iodev_all_channels[i];

    rv = emul_sensor_backend_set_channel(
      emul, ch, &channel_table[ch].expected_values[iteration],
      channel_table[ch].expected_value_shift);
```

Note: some lines dropped for brevity

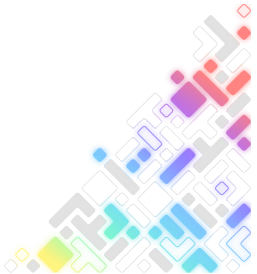
Generic Test - Check Results

- (not shown: read and retrieve values from Sensor API)
- Normalize all the fixed-point numbers in a big `int64_t` container
- Additional check to ensure all requested channels are received

```
	/* Align everything to be a 64-bit Q32.32 number for comparison */
	int64_t expected_shifted =
	  (int64_t)channel_table[ch].expected_values[iteration]
	  << channel_table[ch].expected_value_shift;
	int64_t actual_shifted = (int64_t)q << shift;
	int64_t epsilon_shifted = (int64_t)channel_table[ch].epsilon
	        << channel_table[ch].expected_value_shift;

	zassert_within(expected_shifted, actual_shifted, epsilon_shifted,
	        "Expected %lld, got %lld (shift %d, ch %d, iteration %d/%d, "
	        "Error %lld, Epsilon %lld)",
	        expected_shifted, actual_shifted, shift, ch, iteration + 1,
	        CONFIG_GENERIC_SENSOR_TEST_NUM_EXPECTED_VALS,
	        expected_shifted - actual_shifted, epsilon_shifted);
}
```

Conclusion

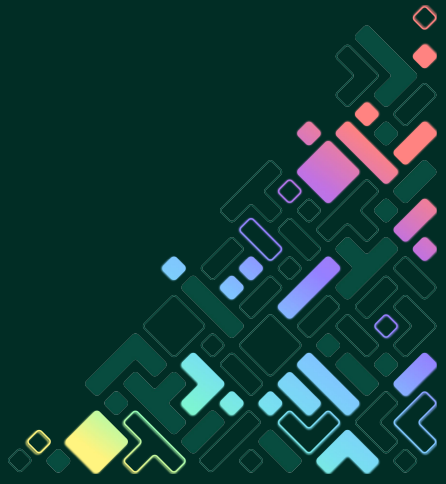
- Tl;dr: Write a backend API implementation for an emulator, get a free test
- Is there a particular sensor important to your project / product?
  - Consider contributing an emulator
  - Time commitment: ~0.5-1 day
  - Prevent regressions!



PRs to explore

- Initial (Generic Test, API, and AKM09918C digital compass impl.)
  - https://github.com/zephyrproject-rtos/zephyr/pull/60394
- ICM42688 motion sensor emul backend API implementation
  - https://github.com/zephyrproject-rtos/zephyr/pull/61051
- SB-TSI (Sideband temperature sensor interface) emul w/ backend API
  - https://github.com/zephyrproject-rtos/zephyr/pull/60818
- Add attribute support to backend API, add BMI160 accel backend API impl.
  - https://github.com/zephyrproject-rtos/zephyr/pull/65278
- Yuval Peress's ZDS 2023 talk on Sensors V2
  - https://eoss2023.sched.com/event/1Leca

Q&A



Zephyr® Project

Developer Summit