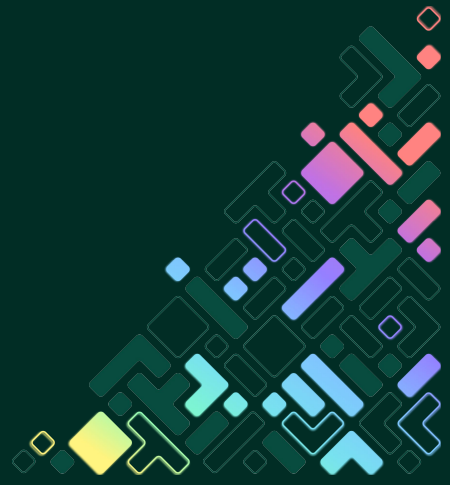
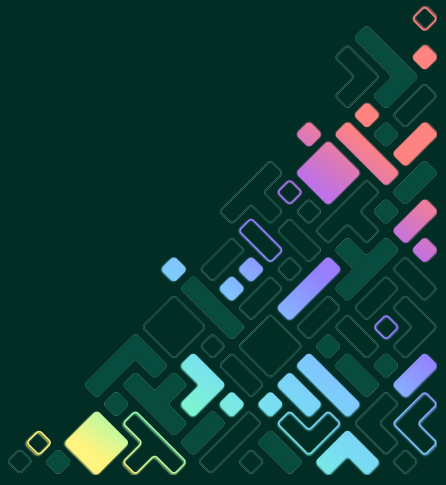
Zephyr® Project
Developer Summit

Unit Testing Memory Mapped Device Drivers

Jeremy Bettis, Google, ChromeOS



Philosophy



Testing pyramid ideals

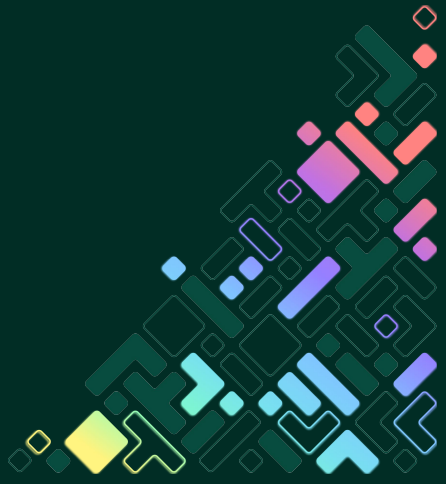
1) Manual testing or automated tests on hardware.
    a) Few, very slow, flaky
    b) Not run often
    c) High confidence
    d) No negative tests
2) Integration tests & QEMU testing.
    a) Some, somewhat slow, flaky
    b) Run in github CI
3) Unit tests.
    a) Lots, fast, stable
    b) Prevent pull-requests from merging

Reality

There are some unit tests. Very few for drivers though.

- It’s too hard
- I don’t have time
- Not worth the effort
- **It’s impossible to intercept memory mapped register accesses**

Case Study



ITE GPIO driver (gpio_ite_it8xxx2_v2)

Wrap all register accesses with a macro

Potential improvement:

SET_EC_REG(reg_addr, value)

READ_EC_REG(reg_addr)

```
/*
 * Macros for hardware registers access.
 */
#define ECREG(x)        (*((volatile unsigned char *)(x)))
#define ECREG_u16(x)    (*((volatile unsigned short *)(x)))
#define ECREG_u32(x)    (*((volatile unsigned long  *)(x)))

BEFORE:
*reg_gpcr = GPCR_PORT_PIN_MODE_TRISTATE;
*value = *reg_gpdmr;

AFTER:
ECREG(reg_gpcr) = GPCR_PORT_PIN_MODE_TRISTATE;
*value = ECREG(reg_gpdmr);
```

ITE GPIO driver (gpio_ite_it8xxx2_v2)

Create a native_sim test.

Enable the driver in the test.

Create a DTS overlay to instantiate it. Use virtual (vnd) or emulated (zephyr) devices for dependencies.

prj.conf:
```
CONFIG_GPIO_ITE_IT8XXX2_V2=y
```

native_sim.overlay:
```
/ {
        intc: interrupt-controller@f03f00 {
                compatible = "vnd,intc";
                ...
        };

        gpioa: gpio@f01601 {
                compatible = "ite,it8xxx2-gpio-v2";
                reg = <0x00f01601 1   /* GPDR (set) */
                       0x00f01618 1   /* GPDMR (get) */
                       0x00f01630 1   /* GPOTR */
                       0x00f01648 1   /* P18SCR */
                       0x00f01660 8>; /* GPCR */
                ...
        };
};
```

ITE GPIO driver (gpio_ite_it8xxx2_v2)

Provide missing headers/functions

Headers in the test override system headers

CMakeLists.txt:

```
zephyr_include_directories(
  include
  ${ZEPHYR_BASE}/soc/ite/ec/common
  ${ZEPHYR_BASE}/soc/ite/ec/it8xxx2
)
```

…/include/zephyr/arch/cpu.h:

```
#include <chip_chipregs.h>
#include <zephyr/arch/posix/arch.h>

int arch_irq_connect_dynamic(unsigned int irq, unsigned int
priority,
          void (*routine)(const void *parameter),
          const void *parameter, uint32_t flags);
int arch_irq_disconnect_dynamic(unsigned int irq, unsigned int
priority,
            void (*routine)(const void *parameter),
            const void *parameter, uint32_t flags);
typedef struct z_thread_stack_element k_thread_stack_t;
```

ITE GPIO driver (gpio_ite_it8xxx2_v2)

Override ECREG to mock out the register access.

tests/drivers/gpio/gpio_ite_it8xxx2_v2/include/chip_chipregs.h:

```
#include <../soc/riscv/riscv-ite/common/chip_chipregs.h>

/*
 * Macros for emulated hardware registers access.
 */
#undef ECREG
#define ECREG(x)        (*((volatile unsigned char
*)fake_ecreg((intptr_t)x)))

unsigned int *fake_ecreg(intptr_t r);
```

ITE GPIO driver (gpio_ite_it8xxx2_v2)

In the test, redirect reads/writes to test variables.

During each test, check the registers for side effects.

```
zassert_true(device_is_ready(gpio_dev));
zassert_ok(gpio_pin_configure(gpio_dev, TEST_PIN,
GPIO_OUTPUT_INACTIVE | GPIO_ACTIVE_LOW));
zexpect_equal(registers.gpotr, 0, "gpotr=%x", registers.gpotr);
zexpect_equal(registers.p18scr, 0);
zexpect_equal(registers.gpcr[TEST_PIN],
GPCR_PORT_PIN_MODE_OUTPUT, "gpcr[%d]=%x",
TEST_PIN, registers.gpcr[TEST_PIN]);
```

tests/drivers/gpio/gpio_ite_it8xxx2_v2/src/main.c:

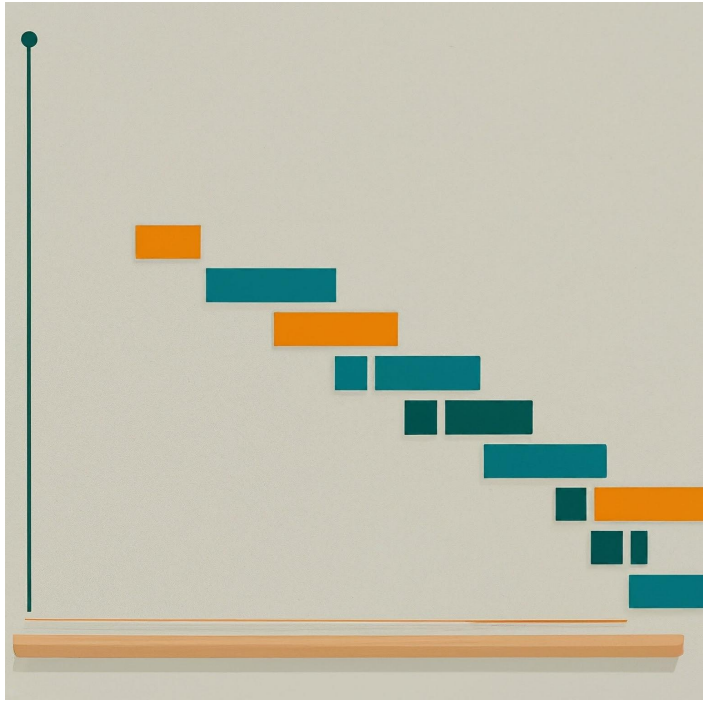
```
static struct {
    uint8_t fake;
    uint8_t gpdmr;
    uint8_t gpdr;
    uint8_t gpotr;
    uint8_t p18scr;
    uint8_t wuemr, wuesr, wubemr;
    uint8_t gpcr[DT_REG_SIZE_BY_IDX(MY_GPIO, 4)];
    bool clear_gpcr_before_read;
} registers;

unsigned int *fake_ecreg(intptr_t r)
{
    switch (r) {
    case DT_REG_ADDR_BY_IDX(MY_GPIO, 0): /* GPDR */
        return (unsigned int *)&registers.gpdr;
    case DT_REG_ADDR_BY_IDX(MY_GPIO, 1): /* GPDMR */
        return (unsigned int *)&registers.gpdmr;
    case DT_REG_ADDR_BY_IDX(MY_GPIO, 2): /* GPOTR */
        return (unsigned int *)&registers.gpotr;
    case DT_REG_ADDR_BY_IDX(MY_GPIO, 3): /* P18SCR */
        return (unsigned int *)&registers.p18scr;
    case DT_PROP_BY_IDX(MY_GPIO, wuc_base, TEST_PIN):
        return (unsigned int *)&registers.wuemr;
    case DT_PROP_BY_IDX(MY_GPIO, wuc_base, TEST_PIN) + 1:
        return (unsigned int *)&registers.wuesr;
    case DT_PROP_BY_IDX(MY_GPIO, wuc_base, TEST_PIN) + 3:
        return (unsigned int *)&registers.wubemr;
    }
    if (r >= DT_REG_ADDR_BY_IDX(MY_GPIO, 4) &&
        r < DT_REG_ADDR_BY_IDX(MY_GPIO, 4) + DT_REG_SIZE_BY_IDX(MY_GPIO, 4)) {
        if (registers.clear_gpcr_before_read) {
            registers.gpcr[r - DT_REG_ADDR_BY_IDX(MY_GPIO, 4)] = 0;
        }
        return (unsigned int *)&registers.gpcr[r - DT_REG_ADDR_BY_IDX(MY_GPIO, 4)];
    }
    zassert_unreachable("Register access: %x", r);
    return (unsigned int *)&registers.fake;
}
```

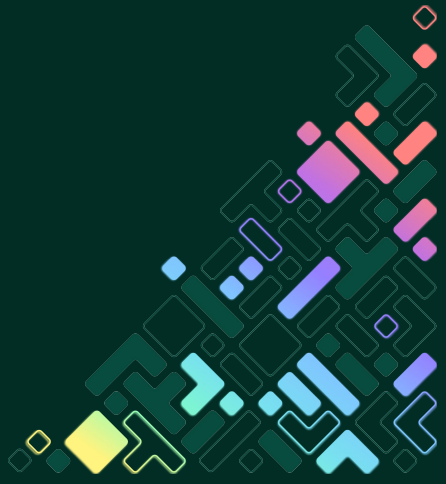
Limitations

Limitations

- If a register doesn’t read the same as the data written to it, the test can’t easily reproduce this.
  - Separate SET_EC_REG/READ_EC_REG macros would help with that.
- These tests risk being “works as coded” tests.
- There is no emulator device for larger integration tests.



Success Story



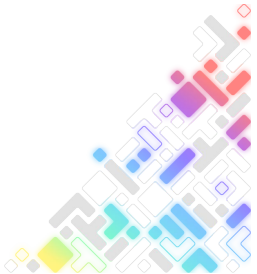
Success Story

New feature added to ITE GPIO driver #66401: Level interrupt triggering

Never tested on hardware, but the unit test had 100% coverage.

Used native_sim, not unit_testing for access to work queues.

Enabled on new Chromebook recently, and it worked perfectly on the actual hardware.



Zephyr® Project

Developer Summit