Zephyr: Button Callbacks

Medical Electrical Equipment (BME590L)

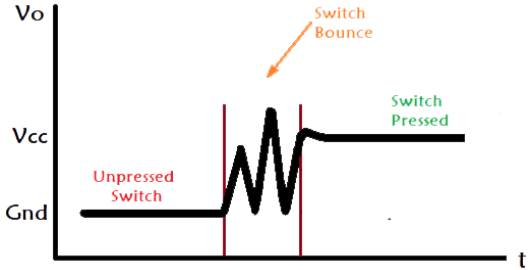
Mark L. Palmeri, M.D., Ph.D.

February 15, 2023

Motivation

What are challenges with detecting button presses?

Switch Bouncing



1

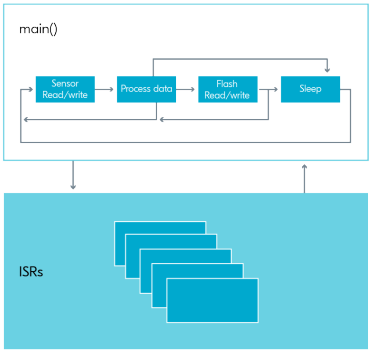
[1] https://circuitdigest.com/electronic-circuits/what-is-switch-bouncing-and-how-to-prevent-it-using-debouncer-circuit

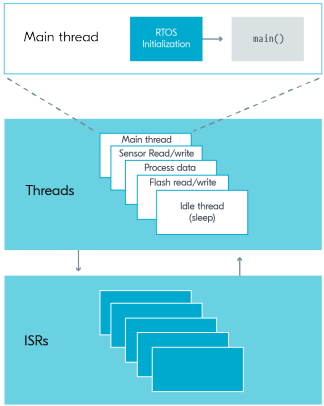
Timing of "read" in main()

```
void main() {
    while (1) {
        // do stuff
        k_msleep(1000);
        sw0_event = command_to_read_digital_pin();
        if (sw0_event) {
            // do something
        }
    }
}
```

- What if you press the button while the code is "sleeping"?

Bare-Metal vs. RTOS



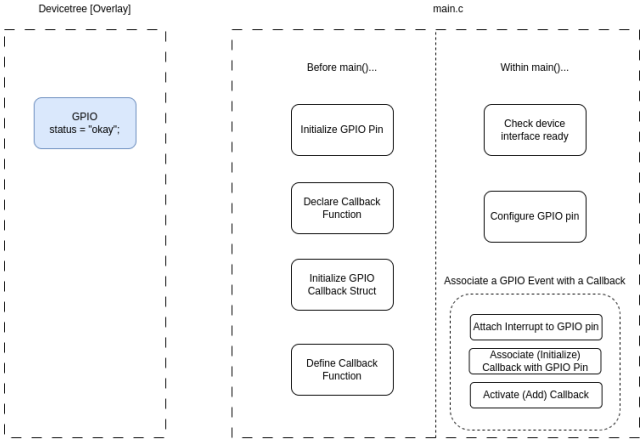


Duke | BME

ISR → Callback Function

- An interrupt can be used to call a "callback" function.
- Need to release the callback / ISR quickly to not paralyze the rest of `main()` / other threads from running, otherwise device is paralyzed from acting.
- Avoid calculations, significant IO, data Tx/Rx, etc.
- Prefer simple actions, like toggling the state of a Boolean variable.

Overview


Devicetree [Overlay]
GPIO
status = "okay";
main.c
Before main()...
Initialize GPIO Pin
Declare Callback Function
Initialize GPIO Callback Struct
Define Callback Function
Within main()...
Check device interface ready
Configure GPIO pin
Associate a GPIO Event with a Callback
Attach Interrupt to GPIO pin
Associate (Initialize) Callback with GPIO Pin
Activate (Add) Callback

Devicetree: gpio-keys

```
/ {
    aliases {
        sw0: &button0;
    }
    buttons {
        compatible = "gpio-keys";
        button0: button_0 {
            gpios = <&gpio0 8 (GPIO_PULL_UP | GPIO_ACTIVE_LOW)>;
            label = "Push button";
        };
    };
};
```

Initialize GPIO struct

```
static const struct gpio_dt_spec sw0 = GPIO_DT_SPEC_GET(DT_ALIAS(sw0), gpios);
```

- Before `main()`
- `GPIO_DT_SPEC_GET`: macro to get GPIO information from the devicetree
- `DT_ALIAS`: reference the pin of interest by an alias
- `gpio_dt_spec`: struct to store all of the information about this GPIO pin

Declare Callback Function

```
void sw0_callback(const struct device *dev, struct gpio_callback *cb, uint32_t
↪  pins);
```

Initialize GPIO Callback Struct

```
static struct gpio_callback sw0_cb;
```

Define Callback Function

```
void sw0_callback(const struct device *dev, struct gpio_callback *cb, uint32_t
↪  pins)
{
    sw0_event = 1;  // conditional statement in main() can now do something based
    ↪  on the event detection
}
```

- The contents of this function should consume minimal resources / time.
- Common action is to toggle the state of a Boolean, the value of which is reset after action is taken in the main code.

Duke | BME

Check if interface is ready

```
if (!device_is_ready(sw0.port)) {
    LOG_ERR("gpio0 interface not ready.");
    return -1;
}
```

Configure GPIO pin

```
int err;
err = gpio_pin_configure_dt(&sw0, GPIO_INPUT);
if (err < 0) {
    LOG_ERR("Cannot configure sw0 pin.");
    return err;
}
```

Associate Callback with GPIO Pin

```
err = gpio_pin_interrupt_configure_dt(&sw0,
↪  GPIO_INT_EDGE_TO_ACTIVE);
if (err < 0) {
    LOG_ERR("Cannot attach callback to sw0.");
}
gpio_init_callback(&sw0_cb, sw0_callback, BIT(sw0.pin));
gpio_add_callback(sw0.port, &sw0_cb);
```

- `GPIO_INT_EDGE_TO_ACTIVE` triggers based on a rising edge from `LOW` to `HIGH`. (More immune to "bounce" in switch/button.)
- `gpio_init_callback()` populates the CB struct with information about the CB function and pin.
- `gpio_add_callback()` associates the CB struct with the port.

Duke | BME

Test for the callback event state in your code...

```
if (sw0_event) {
    do_something_less_trivial();
    sw0_event = 0;
}
```

`do_something_less_trivial()` can consume more resources / take more time than the callback function that was called from the ISR.