**Lesson 4: GPIO**

- 4.1 Introduction
- 4.2 Commands
- 4.3 Kconfig
- 4.4 Exercise

Introduction

In this lesson we'll be covering the way GPIO works in Zephyr. Just the basics:

- set up an input pin
- set up an output pin
- set up an interrupt pin

**GPIO setup**

When setting up any GPIO the following basic steps have to be followed:

1. First you need to look up the devicetree binding for the corresponding GPIO (if you don't know what a devicetree is: see section below).

The devicetree for your particular build can be found at `build/zephyr/zephyr.dts` or for each board in `zephyrproject/zephyr/boards`

For example, the red arrow indicates the device binding to toggle the green led.


2. To use the device binding in our main.c file; we need to use the following defines:

- Devicetree node identifier

#define LED0_NODE DT_ALIAS(led0)
Notice how we used `led0` instead of `green_led` here? This is possible due to the following aliases being defined:

```
aliases {
        led0 = &green_led;
        led1 = &blue_led;
        led2 = &red_led;
        sw0 = &user_button;
};
```

Now we use LED0_NODE to obtain all other (optional) defines:

- Label

#define LED0    DT_GPIO_LABEL(LED0_NODE, gpios)

- Pin

#define PIN DT_GPIO_PIN(LED0_NODE, gpios)

- (Default) flags

#define FLAGS DT_GPIO_FLAGS(LED0_NODE, gpios)
Optionally, instead of using 'defines', you can use `gpio_dt_spec`:
static struct gpio_dt_spec led = GPIO_DT_SPEC_GET_OR(DT_ALIAS(led0), gpios, {0});

static const struct gpio_dt_spec button = GPIO_DT_SPEC_GET_OR(SW0_NODE, gpios, {0});

3. Configure GPIO

- input/output configuration

const struct device *dev;

dev = device_get_binding(LED0);

ret = gpio_pin_configure(dev, PIN, GPIO_INPUT | FLAGS); // For input pin

ret = gpio_pin_configure(dev, PIN, GPIO_OUTPUT | FLAGS); // For output pin

- interrupt configuration

ret = gpio_pin_interrupt_configure(dev, PIN, GPIO_INT_EDGE_RISING | FLAGS); // For rising edge interrupt
Using `gpio_dt_spec`

- input/output configuration

ret = gpio_pin_configure_dt(&led, GPIO_OUTPUT);
ret = gpio_pin_configure_dt(&button, GPIO_INPUT);

- interrupt configuration

gpio_pin_interrupt_configure_dt(&button, GPIO_INT_EDGE_RISING);

4. Use GPIO through dedicated functions (set, read, toggle,...)

- set

gpio_pin_set(dev, PIN, led_state);

- read

button_state = gpio_pin_get(dev, PIN);

- toggle

gpio_pin_toggle(dev, PIN);
Using `gpio_dt_spec`

- set

int val = gpio_pin_set_dt(&led, val);

- read

int val = gpio_pin_get_dt(&button);

- toggle

gpio_pin_toggle_dt(&led);

**Devicetree**

A device tree is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the peripherals. Wikipedia
In short: the devicetree will help Zephyr locate all the components of your particular SoC/Board and in this way it will be able to work regardless of the underlying hardware. That also means if we want to access anything on the board from within Zephyr, we'll need to know the binding of that particular component in our devicetree. For example, if we want to blink an LED on the board, we'll first have to determine how that LED is called in the devicetree. If you want to learn more about devicetree, I'd recommend watching this video, which explains it pretty well. (devicetree is a concept borrowed from Linux)

The devicetree for the dev board (nucleo-f756zg) can be be found at `zephyrproject/zephyr/boards/arm/nucleo_f756zg` and for the microcontroller itself (stm32f756) at `zephyrproject/zephyr/dts/arm/st/f7/stm32f756Xg.dtsi`. If you study the devicetree files, you'll notice that the final devicetree is a combination of multiple files; the way this works is that each successive devicetree file gets laid over the previous one thereby forming a final devicetree (to be found in `build/zephyr/zephyr.dts`) this one should match your particular dev board and microcontroller.

Useful API pages:

- GPIO API
- Devicetree
- Device Driver Model

**Functions table**

| Function | Description |
|---|---|
| gpio_pin_configure | Configure a single pin |
| gpio_pin_interrupt_configure | Configure pin interrupt |
| gpio_pin_get | Get logical level of an input pin - taking into account the GPIO_ACTIVE_LOW flag. If pin is configured as Active High, a low physical level will be interpreted as logical value 0. If pin is configured as Active Low, a low physical level will be interpreted as logical value 1. |
| gpio_pin_set | Set logical level of an output pin taking into account GPIO_ACTIVE_LOW flag. Value 0 sets the pin in logical 0 / inactive state. Any value other than 0 sets the pin in logical 1 / active state. If pin is configured as Active High, the default, setting it in inactive state will force the pin to a low physical level. If pin is configured as Active Low, setting it in inactive state will force the pin to a high physical level. |
| gpio_pin_toggle | Toggle pin level |

**Callback functions: (not covered in exercises)**

| Function | Description |
|---|---|
| gpio_init_callback | Helper to initialize a struct gpio_callback properly |
| gpio_add_callback | Add an application callback |
| gpio_remove_callback | Remove an application callback |

**Device Driver Model**

| Function | Description |
|---|---|
| device_get_binding | Retrieve the device structure for a driver by name. Device objects are created via the DEVICE_DEFINE() macro and placed in memory by the linker. If a driver needs to bind to another driver it can use this function to retrieve the device structure of the lower level driver by the name the driver exposes to the system. |

**Flags table**

**Input/Output configuration options:**

| Flags | Description |
|---|---|
| GPIO_INPUT | Enables pin as input |
| GPIO_OUTPUT | Enables pin as output, no change to the output state |
| GPIO_DISCONNECTED | Disables pin for both input and output. |
| GPIO_OUTPUT_LOW | Configures GPIO pin as output and initializes it to a low state. |
| GPIO_OUTPUT_HIGH | Configures GPIO pin as output and initializes it to a high state. |
| GPIO_OUTPUT_INACTIVE | Configures GPIO pin as output and initializes it to a logic 0. |
| GPIO_OUTPUT_ACTIVE | Configures GPIO pin as output and initializes it to a logic 1. |

**GPIO interrupt configuration flags**

| Flags | Description |
|---|---|
| GPIO_INT_DISABLE | Disables GPIO pin interrupt. |
| GPIO_INT_EDGE_RISING | Configures GPIO interrupt to be triggered on pin rising edge and enables it. |
| GPIO_INT_EDGE_FALLING | Configures GPIO interrupt to be triggered on pin falling edge and enables it. |

| Flags | Description |
|---|---|
| GPIO_INT_EDGE_BOTH | Configures GPIO interrupt to be triggered on pin rising or falling edge and enables it. |
| GPIO_INT_LEVEL_LOW | Configures GPIO interrupt to be triggered on pin physical level low and enables it. |
| GPIO_INT_LEVEL_HIGH | Configures GPIO interrupt to be triggered on pin physical level high and enables it. |
| GPIO_INT_EDGE_TO_INACTIVE | Configures GPIO interrupt to be triggered on pin state change to logical level 0 and enables it. |
| GPIO_INT_EDGE_TO_ACTIVE | Configures GPIO interrupt to be triggered on pin state change to logical level 1 and enables it. |
| GPIO_INT_LEVEL_INACTIVE | Configures GPIO interrupt to be triggered on pin logical level 0 and enables it. |
| GPIO_INT_LEVEL_ACTIVE | Configures GPIO interrupt to be triggered on pin logical level 1 and enables it. |

**GPIO pin active level flags**

| Flags | Description |
|---|---|
| GPIO_ACTIVE_LOW | GPIO pin is active (has logical value ‘1’) in low state. |
| GPIO_ACTIVE_HIGH | GPIO pin is active (has logical value ‘1’) in high state. |

**Defines**

| Flags | Description |
|---|---|
| GPIO_INT_DEBOUNCE | Enable GPIO pin debounce |
| GPIO_DT_SPEC_GET_BY_IDX | This returns a static initializer for a gpio_dt_spec structure given a devicetree node identifier, a property specifying a GPIO and an index. |

| Flags | Description |
|---|---|
| DT_ALIAS | Get a node identifier from /aliases. |
| DT_GPIO_LABEL | Get a label property from a gpio phandle-array property (at index 0) |
| DT_GPIO_PIN | Get a GPIO specifier’s pin cell (at index 0) |
| DT_GPIO_FLAGS | Get a GPIO specifier’s flags cell (at index 0) |

Additional flag categories can be found here:

- GPIO drive strength flags
- GPIO pin drive flags
- GPIO pin bias flags
- GPIO pin voltage flags

**Blinky**

GPIO-output

Make a 1 Hz blinky program.

solution: `exercises/gpio/blinky`

**Button**

GPIO-input

Use button to turn LED on or off.

solution: `exercises/gpio/button`

**2 LEDs**

Threads + GPIO

Blinky for 2 LED at different frequencies, using different threads.

solution: `exercises/gpio/two-leds`