Dedicated GPIO

[中文]

Overview

The dedicated GPIO is designed for CPU interaction with GPIO matrix and IO MUX. Any GPIO that is configured as "dedicated" can be access by CPU instructions directly, which makes it easy to achieve a high GPIO flip speed, and simulate serial/parallel interface in a bit-banging way. As toggling a GPIO in this "CPU Dedicated" way costs few overhead, it would be great for cases like performance measurement using an oscilloscope.

Create/Destroy GPIO Bundle

A GPIO bundle is a group of GPIOs, which can be manipulated at the same time in one CPU cycle. The maximal number of GPIOs that a bundle can contain is limited by each CPU. What's more, the GPIO bundle has a strong relevance to the CPU which it derives from. **Any operations on the GPIO bundle should be put inside a task which is running on the same CPU core to the GPIO bundle belongs to.** Likewise, only those ISRs who are installed on the same CPU core are allowed to do operations on that GPIO bundle.

> **Note**
>
> Dedicated GPIO is more of a CPU peripheral, so it has a strong relationship with CPU core. It's highly recommended to install and operate GPIO bundle in a pin-to-core task. For example, if GPIOA is connected to CPU0, and the dedicated GPIO instruction is issued from CPU1, then it's impossible to control GPIOA.

To install a GPIO bundle, one needs to call `dedic_gpio_new_bundle()` to allocate the software resources and connect the dedicated channels to user selected GPIOs. Configurations for a GPIO bundle are covered in `dedic_gpio_bundle_config_t` structure:

- `gpio_array`: An array that contains GPIO number.
- `array_size`: Element number of `gpio_array`.
- `flags`: Extra flags to control the behavior of GPIO Bundle.

- `in_en` and `out_en` are used to select whether to enable the input and output function (note, they can be enabled together).
- `in_invert` and `out_invert` are used to select whether to invert the GPIO signal.

The following code shows how to install a output only GPIO bundle:

```
// configure GPIO
const int bundleA_gpios[] = {0, 1};
gpio_config_t io_conf = {
    .mode = GPIO_MODE_OUTPUT,
};
for (int i = 0; i < sizeof(bundleA_gpios) / sizeof(bundleA_gpios[0]); i++) {
    io_conf.pin_bit_mask = 1ULL << bundleA_gpios[i];
    gpio_config(&io_conf);
}
// Create bundleA, output only
dedic_gpio_bundle_handle_t bundleA = NULL;
dedic_gpio_bundle_config_t bundleA_config = {
    .gpio_array = bundleA_gpios,
    .array_size = sizeof(bundleA_gpios) / sizeof(bundleA_gpios[0]),
    .flags = {
        .out_en = 1,
    },
};
ESP_ERROR_CHECK(dedic_gpio_new_bundle(&bundleA_config, &bundleA));
```

To uninstall the GPIO bundle, one needs to call `dedic_gpio_del_bundle()`.

Note

`dedic_gpio_new_bundle()` doesn't cover any GPIO pad configuration (e.g. pull up/down, drive ability, output/input enable), so before installing a dedicated GPIO bundle, you have to configure the GPIO separately using GPIO driver API (e.g. `gpio_config()`). For more information about GPIO driver, please refer to GPIO API Reference.

GPIO Bundle Operations

Operations	Functions
Write to GPIOs in the bundle by mask	`dedic_gpio_bundle_write()`
Read the value that output from the given GPIO bundle	`dedic_gpio_bundle_read_out()`
Read the value that input to the given GPIO bundle	`dedic_gpio_bundle_read_in()`

Note

Using the above functions might not get a high GPIO flip speed because of the overhead of function calls and the bit operations involved inside. Users can try Manipulate GPIOs by Writing Assembly Code instead to reduce the overhead but should take care of the thread safety by themselves.

Manipulate GPIOs by Writing Assembly Code

For advanced users, they can always manipulate the GPIOs by writing assembly code or invoking CPU Low Level APIs. The usual procedure could be:

1. Allocate a GPIO bundle: `dedic_gpio_new_bundle()`
2. Query the mask occupied by that bundle: `dedic_gpio_get_out_mask()` or/and `dedic_gpio_get_in_mask()`
3. Call CPU LL apis (e.g. *dedic_gpio_cpu_ll_write_mask*) or write assembly code with that mask
4. The fasted way of toggling IO is to use the dedicated "set/clear" instructions:
    - Set bits of GPIO: `set_bit_gpio_out imm[7:0]`
    - Clear bits of GPIO: `clr_bit_gpio_out imm[7:0]`
    - Note: Immediate value width depends on the number of dedicated GPIO channels

For details of supported dedicated GPIO instructions, please refer to *ESP32-S3 Technical Reference Manual > Processor Instruction Extensions (PIE) (to be added later)* [PDF].

Some of the dedicated CPU instructions are also wrapped inside `hal/dedic_gpio_cpu_ll.h` as helper inline functions.

Note

Writing assembly code in application could make your code hard to port between targets, because those customized instructions are not guaranteed to remain the same format on different targets.

API Reference

Header File

- components/driver/include/driver/dedic_gpio.h

Functions

esp_err_t **dedic_gpio_get_out_mask**(dedic_gpio_bundle_handle_t bundle, uint32_t *mask)

Get allocated channel mask.

Note

Each bundle should have at least one mask (in or/and out), based on bundle configuration.

Note

With the returned mask, user can directly invoke LL function like "dedic_gpio_cpu_ll_write_mask" or write assembly code with dedicated GPIO instructions, to get better performance on GPIO manipulation.

Parameters:

- **bundle** – **[in]** Handle of GPIO bundle that returned from "dedic_gpio_new_bundle"
- **mask** – **[out]** Returned mask value for on specific direction (in or out)

Returns:

- ESP_OK: Get channel mask successfully
- ESP_ERR_INVALID_ARG: Get channel mask failed because of invalid argument
- ESP_FAIL: Get channel mask failed because of other error

---

esp_err_t **dedic_gpio_get_in_mask**(dedic_gpio_bundle_handle_t bundle, uint32_t *mask)

---

esp_err_t **dedic_gpio_new_bundle**(*const* dedic_gpio_bundle_config_t *config, dedic_gpio_bundle_handle_t *ret_bundle)

Create GPIO bundle and return the handle.

Note

One has to enable at least input or output mode in "config" parameter.

Parameters:

- **config** – **[in]** Configuration of GPIO bundle
- **ret_bundle** – **[out]** Returned handle of the new created GPIO bundle

Returns:

- ESP_OK: Create GPIO bundle successfully
- ESP_ERR_INVALID_ARG: Create GPIO bundle failed because of invalid argument
- ESP_ERR_NO_MEM: Create GPIO bundle failed because of no capable memory
- ESP_ERR_NOT_FOUND: Create GPIO bundle failed because of no enough continuous dedicated channels

- ESP_FAIL: Create GPIO bundle failed because of other error

---

esp_err_t **dedic_gpio_del_bundle**(dedic_gpio_bundle_handle_t bundle)

Destory GPIO bundle.

Parameters: **bundle** – **[in]** Handle of GPIO bundle that returned from “dedic_gpio_new_bundle”

Returns:

- ESP_OK: Destory GPIO bundle successfully
- ESP_ERR_INVALID_ARG: Destory GPIO bundle failed because of invalid argument
- ESP_FAIL: Destory GPIO bundle failed because of other error

---

void **dedic_gpio_bundle_write**(dedic_gpio_bundle_handle_t bundle, uint32_t mask, uint32_t value)

Write value to GPIO bundle.

Note

The mask is seen from the view of GPIO bundle. For example, bundleA contains [GPIO10, GPIO12, GPIO17], to set GPIO17 individually, the mask should be 0x04.

Note

For performance reasons, this function doesn’t check the validity of any parameters, and is placed in IRAM.

Parameters:

- **bundle** – **[in]** Handle of GPIO bundle that returned from “dedic_gpio_new_bundle”
- **mask** – **[in]** Mask of the GPIOs to be written in the given bundle
- **value** – **[in]** Value to write to given GPIO bundle, low bit represents low member in the bundle

---

uint32_t **dedic_gpio_bundle_read_out**(dedic_gpio_bundle_handle_t bundle)

Read the value that output from the given GPIO bundle.

Note

For performance reasons, this function doesn’t check the validity of any parameters, and is placed in IRAM.

Parameters: **bundle** – **[in]** Handle of GPIO bundle that returned from “dedic_gpio_new_bundle”

Returns: Value that output from the GPIO bundle, low bit represents low member in the bundle

---

uint32_t **dedic_gpio_bundle_read_in**(dedic_gpio_bundle_handle_t bundle)

Read the value that input to the given GPIO bundle.

Note

For performance reasons, this function doesn’t check the validity of any parameters, and is placed in IRAM.

Parameters: **bundle** – **[in]** Handle of GPIO bundle that returned from “dedic_gpio_new_bundle”

Returns: Value that input to the GPIO bundle, low bit represents low member in the bundle

Structures

---

struct **dedic_gpio_bundle_config_t**

Type of Dedicated GPIO bundle configuration.

Public Members

const int ***gpio_array**

Array of GPIO numbers, gpio_array[0] ~ gpio_array[size-1] <=> low_dedic_channel_num ~ high_dedic_channel_num

size_t **array_size**

Number of GPIOs in gpio_array

unsigned int **in_en**

Enable input

unsigned int **in_invert**

Invert input signal

unsigned int **out_en**

Enable output

unsigned int **out_invert**

Invert output signal

struct dedic_gpio_bundle_config_t::[anonymous] **flags**

Flags to control specific behaviour of GPIO bundle

Type Definitions

typedef struct dedic_gpio_bundle_t ***dedic_gpio_bundle_handle_t**

Type of Dedicated GPIO bundle.