General Purpose Input/Output (GPIO)

Contents:

Core

struct **gpio_irq_chip**

GPIO interrupt controller

Definition

```
struct gpio_irq_chip {
  struct irq_chip *chip;
  struct irq_domain *domain;
  const struct irq_domain_ops *domain_ops;
#ifdef CONFIG_IRQ_DOMAIN_HIERARCHY;
  struct fwnode_handle *fwnode;
  struct irq_domain *parent_domain;
  int (*child_to_parent_hwirq)(struct gpio_chip *gc,unsigned int child_hwirq,unsigned int
child_type,unsigned int *parent_hwirq, unsigned int *parent_type);
  int (*populate_parent_alloc_arg)(struct gpio_chip *gc,union gpio_irq_fwspec *fwspec,unsigned int
parent_hwirq, unsigned int parent_type);
  unsigned int (*child_offset_to_irq)(struct gpio_chip *gc, unsigned int pin);
  struct irq_domain_ops child_irq_domain_ops;
#endif;
  irq_flow_handler_t handler;
  unsigned int default_type;
  struct lock_class_key *lock_key;
  struct lock_class_key *request_key;
  irq_flow_handler_t parent_handler;
  union {
    void *parent_handler_data;
    void **parent_handler_data_array;
  };
  unsigned int num_parents;
  unsigned int *parents;
  unsigned int *map;
  bool threaded;
  bool per_parent_data;
  bool initialized;
  int (*init_hw)(struct gpio_chip *gc);
  void (*init_valid_mask)(struct gpio_chip *gc,unsigned long *valid_mask, unsigned int ngpios);
  unsigned long *valid_mask;
  unsigned int first;
  void (*irq_enable)(struct irq_data *data);
  void (*irq_disable)(struct irq_data *data);
  void (*irq_unmask)(struct irq_data *data);
  void (*irq_mask)(struct irq_data *data);
};
```

Members

`chip`

GPIO IRQ chip implementation, provided by GPIO driver.

`domain`

Interrupt translation domain; responsible for mapping between GPIO hwirq number and Linux IRQ number.

`domain_ops`

Table of interrupt domain operations for this IRQ chip.

`fwnode`

Firmware node corresponding to this gpiochip/irqchip, necessary for hierarchical irqdomain support.

`parent_domain`

If non-NULL, will be set as the parent of this GPIO interrupt controller's IRQ domain to establish a hierarchical interrupt domain. The presence of this will activate the hierarchical interrupt support.

`child_to_parent_hwirq`

This callback translates a child hardware IRQ offset to a parent hardware IRQ offset on a hierarchical interrupt chip. The child hardware IRQs correspond to the GPIO index 0..ngpio-1 (see the ngpio field of `struct gpio_chip`) and the corresponding parent hardware IRQ and type (such as IRQ_TYPE_*) shall be returned by the driver. The driver can calculate this from an offset or using a lookup table or whatever method is best for this chip. Return 0 on successful translation in the driver.

If some ranges of hardware IRQs do not have a corresponding parent HWIRQ, return -EINVAL, but also make sure to fill in **valid_mask** and **need_valid_mask** to make these GPIO lines unavailable for translation.

`populate_parent_alloc_arg`

This optional callback allocates and populates the specific struct for the parent's IRQ domain. If this is not specified, then `gpiochip_populate_parent_fwspec_twocell` will be used. A four-cell variant named `gpiochip_populate_parent_fwspec_fourcell` is also available.

`child_offset_to_irq`

This optional callback is used to translate the child's GPIO line offset on the GPIO chip to an IRQ number for the GPIO to_irq() callback. If this is not specified, then a default callback will be provided that returns the line offset.

`child_irq_domain_ops`

The IRQ domain operations that will be used for this GPIO IRQ chip. If no operations are provided, then default callbacks will be populated to setup the IRQ hierarchy. Some drivers need to supply their own translate function.

`handler`

The IRQ handler to use (often a predefined IRQ core function) for GPIO IRQs, provided by GPIO driver.

`default_type`

Default IRQ triggering type applied during GPIO driver initialization, provided by GPIO driver.

`lock_key`

Per GPIO IRQ chip lockdep class for IRQ lock.

`request_key`

Per GPIO IRQ chip lockdep class for IRQ request.

`parent_handler`

The interrupt handler for the GPIO chip's parent interrupts, may be NULL if the parent interrupts are nested rather than cascaded.

`{unnamed_union}`

anonymous

`parent_handler_data`

If **per_parent_data** is false, **parent_handler_data** is a single pointer used as the data associated with every parent interrupt.

`parent_handler_data_array`

If **per_parent_data** is true, **parent_handler_data_array** is an array of **num_parents** pointers, and is used to associate different data for each parent. This cannot be NULL if **per_parent_data** is true.

`num_parents`

The number of interrupt parents of a GPIO chip.

`parents`

A list of interrupt parents of a GPIO chip. This is owned by the driver, so the core will only reference this list, not modify it.

`map`

A list of interrupt parents for each line of a GPIO chip.

`threaded`

True if set the interrupt handling uses nested threads.

`per_parent_data`

True if parent_handler_data_array describes a **num_parents** sized array to be used as parent data.

`initialized`

Flag to track GPIO chip irq member's initialization. This flag will make sure GPIO chip irq members are not used before they are initialized.

`init_hw`

optional routine to initialize hardware before an IRQ chip will be added. This is quite useful when a particular driver wants to clear IRQ related registers in order to avoid undesired events.

`init_valid_mask`

optional routine to initialize **valid_mask**, to be used if not all GPIO lines are valid interrupts. Sometimes some lines just cannot fire interrupts, and this routine, when defined, is passed a bitmap in "valid_mask" and it will have ngpios bits from 0..(ngpios-1) set to "1" as in valid. The callback can then directly set some bits to "0" if they cannot be used for interrupts.

`valid_mask`

If not `NULL`, holds bitmask of GPIOs which are valid to be included in IRQ domain of the chip.

`first`

Required for static IRQ allocation. If set, irq_domain_add_simple() will allocate and map all IRQs during initialization.

`irq_enable`

Store old irq_chip irq_enable callback

`irq_disable`

Store old irq_chip irq_disable callback

`irq_unmask`

Store old irq_chip irq_unmask callback

`irq_mask`

Store old irq_chip irq_mask callback

struct `gpio_chip`

abstract a GPIO controller

Definition

```
struct gpio_chip {
  const char              *label;
  struct gpio_device      *gpiodev;
  struct device           *parent;
  struct fwnode_handle    *fwnode;
  struct module           *owner;
  int (*request)(struct gpio_chip *gc, unsigned int offset);
  void (*free)(struct gpio_chip *gc, unsigned int offset);
  int (*get_direction)(struct gpio_chip *gc, unsigned int offset);
  int (*direction_input)(struct gpio_chip *gc, unsigned int offset);
  int (*direction_output)(struct gpio_chip *gc, unsigned int offset, int value);
  int (*get)(struct gpio_chip *gc, unsigned int offset);
  int (*get_multiple)(struct gpio_chip *gc,unsigned long *mask, unsigned long *bits);
  void (*set)(struct gpio_chip *gc, unsigned int offset, int value);
  void (*set_multiple)(struct gpio_chip *gc,unsigned long *mask, unsigned long *bits);
  int (*set_config)(struct gpio_chip *gc,unsigned int offset, unsigned long config);
  int (*to_irq)(struct gpio_chip *gc, unsigned int offset);
  void (*dbg_show)(struct seq_file *s, struct gpio_chip *gc);
  int (*init_valid_mask)(struct gpio_chip *gc,unsigned long *valid_mask, unsigned int ngpios);
  int (*add_pin_ranges)(struct gpio_chip *gc);
  int (*en_hw_timestamp)(struct gpio_chip *gc,u32 offset, unsigned long flags);
  int (*dis_hw_timestamp)(struct gpio_chip *gc,u32 offset, unsigned long flags);
  int base;
  u16 ngpio;
  u16 offset;
  const char              *const *names;
  bool can_sleep;
#if IS_ENABLED(CONFIG_GPIO_GENERIC);
  unsigned long (*read_reg)(void __iomem *reg);
  void (*write_reg)(void __iomem *reg, unsigned long data);
  bool be_bits;
  void __iomem *reg_dat;
  void __iomem *reg_set;
  void __iomem *reg_clr;
  void __iomem *reg_dir_out;
  void __iomem *reg_dir_in;
  bool bgpio_dir_unreadable;
  int bgpio_bits;
  raw_spinlock_t bgpio_lock;
  unsigned long bgpio_data;
  unsigned long bgpio_dir;
#endif ;
#ifdef CONFIG_GPIOLIB_IRQCHIP;
  struct gpio_irq_chip irq;
#endif ;
  unsigned long *valid_mask;
#if defined(CONFIG_OF_GPIO);
  struct device_node *of_node;
  unsigned int of_gpio_n_cells;
  int (*of_xlate)(struct gpio_chip *gc, const struct of_phandle_args *gpiospec, u32 *flags);
  int (*of_gpio_ranges_fallback)(struct gpio_chip *gc, struct device_node *np);
#endif ;
};
```

Members

`label`

a functional name for the GPIO device, such as a part number or the name of the SoC IP-block implementing it.

`gpiodev`

the internal state holder, opaque struct

`parent`

optional parent device providing the GPIOs

`fwnode`

optional fwnode providing this controller’s properties

`owner`

helps prevent removal of modules exporting active GPIOs

`request`

optional hook for chip-specific activation, such as enabling module power and clock; may sleep

`free`

optional hook for chip-specific deactivation, such as disabling module power and clock; may sleep

`get_direction`

returns direction for signal “offset”, 0=out, 1=in, (same as GPIO_LINE_DIRECTION_OUT / GPIO_LINE_DIRECTION_IN), or negative error. It is recommended to always implement this function, even on input-only or output-only gpio chips.

`direction_input`

configures signal “offset” as input, or returns error This can be omitted on input-only or output-only gpio chips.

`direction_output`

configures signal “offset” as output, or returns error This can be omitted on input-only or output-only gpio chips.

`get`

returns value for signal “offset”, 0=low, 1=high, or negative error

`get_multiple`

reads values for multiple signals defined by “mask” and stores them in “bits”, returns 0 on success or negative error

`set`

assigns output value for signal “offset”

`set_multiple`

assigns output values for multiple signals defined by “mask”

`set_config`

optional hook for all kinds of settings. Uses the same packed config format as generic pinconf.

`to_irq`

optional hook supporting non-static gpio_to_irq() mappings; implementation may not sleep

`dbg_show`

optional routine to show contents in debugfs; default code will be used when this is omitted, but custom code can show extra state (such as pullup/pulldown configuration).

`init_valid_mask`

optional routine to initialize **valid_mask**, to be used if not all GPIOs are valid.

`add_pin_ranges`

optional routine to initialize pin ranges, to be used when requires special mapping of the pins that provides GPIO functionality. It is called after adding GPIO chip and before adding IRQ chip.

`en_hw_timestamp`

Dependent on GPIO chip, an optional routine to enable hardware timestamp.

`dis_hw_timestamp`

Dependent on GPIO chip, an optional routine to disable hardware timestamp.

`base`

identifies the first GPIO number handled by this chip; or, if negative during registration, requests dynamic ID allocation. DEPRECATION: providing anything non-negative and nailing the base offset of GPIO chips is deprecated. Please pass -1 as base to let gpiolib select the chip base in all possible cases. We want to get rid of the static GPIO number space in the long run.

`ngpio`

the number of GPIOs handled by this controller; the last GPIO handled is (base + ngpio - 1).

`offset`

when multiple gpio chips belong to the same device this can be used as offset within the device so friendly names can be properly assigned.

`names`

if set, must be an array of strings to use as alternative names for the GPIOs in this chip. Any entry in the array may be NULL if there is no alias for the GPIO, however the array must be **ngpio** entries long. A name can include a single printk format specifier for an unsigned int. It is substituted by the actual number of the gpio.

`can_sleep`

flag must be set iff get()/set() methods sleep, as they must while accessing GPIO expander chips over I2C or SPI. This implies that if the chip supports IRQs, these IRQs need to be threaded as the chip access may sleep when e.g. reading out the IRQ status registers.

`read_reg`

reader function for generic GPIO

`write_reg`

writer function for generic GPIO

`be_bits`

if the generic GPIO has big endian bit order (bit 31 is representing line 0, bit 30 is line 1 … bit 0 is line 31) this is set to true by the generic GPIO core. It is for internal housekeeping only.

`reg_dat`

data (in) register for generic GPIO

`reg_set`

output set register (out=high) for generic GPIO

`reg_clr`

output clear register (out=low) for generic GPIO

`reg_dir_out`

direction out setting register for generic GPIO

`reg_dir_in`

direction in setting register for generic GPIO

`bgpio_dir_unreadable`

indicates that the direction register(s) cannot be read and we need to rely on out internal state tracking.

`bgpio_bits`

number of register bits used for a generic GPIO i.e. <register width> * 8

`bgpio_lock`

used to lock chip->bgpio_data. Also, this is needed to keep shadowed and real data registers writes together.

`bgpio_data`

shadowed data register for generic GPIO to clear/set bits safely.

`bgpio_dir`

shadowed direction register for generic GPIO to clear/set direction safely. A “1” in this word means the line is set as output.

`irq`

Integrates interrupt chip functionality with the GPIO chip. Can be used to handle IRQs for most practical cases.

`valid_mask`

If not `NULL`, holds bitmask of GPIOs which are valid to be used from the chip.

`of_node`

Pointer to a device tree node representing this GPIO controller.

`of_gpio_n_cells`

Number of cells used to form the GPIO specifier.

`of_xlate`

Callback to translate a device tree GPIO specifier into a chip- relative GPIO number and flags.

`of_gpio_ranges_fallback`

Optional hook for the case that no gpio-ranges property is defined within the device tree node “np” (usually DT before introduction of gpio-ranges). So this callback is helpful to provide the necessary backward compatibility for the pin ranges.

Description

A gpio_chip can help platforms abstract various sources of GPIOs so they can all be accessed through a common programming interface. Example sources would be SOC controllers, FPGAs, multifunction chips, dedicated GPIO expanders, and so on.

Each chip controls a number of signals, identified in method calls by “offset” values in the range 0..(**ngpio** - 1). When those signals are referenced through calls like gpio_get_value(gpio), the offset is calculated by subtracting **base** from the gpio number.

`for_each_requested_gpio_in_range`

`for_each_requested_gpio_in_range (chip, i, base, size, label)`

iterates over requested GPIOs in a given range

Parameters

`chip`

the chip to query

`i`

loop variable

`base`

first GPIO in the range

`size`

amount of GPIOs to check starting from **base**

`label`

label of current GPIO

`gpiochip_add_data`

`gpiochip_add_data (gc, data)`

register a gpio_chip

Parameters

`gc`

the chip to register, with gc->base initialized

`data`

driver-private data associated with this chip

Context

potentially before irqs will work

Description

When `gpiochip_add_data()` is called very early during boot, so that GPIOs can be freely used, the gc->parent device must be registered before the gpio framework's arch_initcall(). Otherwise sysfs initialization for GPIOs will fail rudely.

`gpiochip_add_data()` must only be called after gpiolib initialization, i.e. after core_initcall().

If gc->base is negative, this requests dynamic assignment of a range of valid GPIOs.

Return

A negative errno if the chip can’t be registered, such as because the gc->base is invalid or already associated with a different chip. Otherwise it returns zero as a success code.

struct **gpio_pin_range**

pin range controlled by a gpio chip

Definition

```
struct gpio_pin_range {
  struct list_head node;
  struct pinctrl_dev *pctldev;
  struct pinctrl_gpio_range range;
};
```

Members

`node`

list for maintaining set of pin ranges, used internally

`pctldev`

pinctrl device which handles corresponding pins

`range`

actual range of pins controlled by a gpio controller

struct gpio_desc ***gpio_to_desc**(unsigned gpio)

Convert a GPIO number to its descriptor

Parameters

`unsigned gpio`

global GPIO number

Return

The GPIO descriptor associated with the given GPIO, or `NULL` if no GPIO with the given number exists in the system.

struct gpio_desc ***gpiochip_get_desc**(*struct* gpio_chip *gc, unsigned int hwnum)

get the GPIO descriptor corresponding to the given hardware number for this chip

Parameters

`struct gpio_chip *gc`

GPIO chip

`unsigned int hwnum`

hardware number of the GPIO for this chip

Return

A pointer to the GPIO descriptor or `ERR_PTR(-EINVAL)` if no GPIO exists in the given chip for the specified hardware number.

---

int **`desc_to_gpio`**(*const struct* gpio_desc *desc)

convert a GPIO descriptor to the integer namespace

Parameters

`const struct gpio_desc *desc`

GPIO descriptor

Description

This should disappear in the future but is needed since we still use GPIO numbers for error messages and sysfs nodes.

Return

The global GPIO number for the GPIO specified by its descriptor.

---

struct gpio_chip ***`gpiod_to_chip`**(*const struct* gpio_desc *desc)

Return the GPIO chip to which a GPIO descriptor belongs

Parameters

`const struct gpio_desc *desc`

descriptor to return the chip of

---

int **`gpiod_get_direction`**(*struct* gpio_desc *desc)

return the current direction of a GPIO

Parameters

`struct gpio_desc *desc`

GPIO to get the direction of

Description

Returns 0 for output, 1 for input, or an error code in case of error.

This function may sleep if `gpiod_cansleep()` is true.

---

void ***gpiochip_get_data**(*struct* gpio_chip *gc)

get per-subdriver data for the chip

Parameters

`struct gpio_chip *gc`

GPIO chip

Return

The per-subdriver data for the chip.

---

void **gpiochip_remove**(*struct* gpio_chip *gc)

unregister a gpio_chip

Parameters

`struct gpio_chip *gc`

the chip to unregister

Description

A gpio_chip with any GPIOs still requested may not be removed.

---

struct gpio_chip ***gpiochip_find**(void *data, int (*match)(*struct* gpio_chip *gc, void *data))

iterator for locating a specific gpio_chip

Parameters

`void *data`

data to pass to match function

`int (*match)(struct gpio_chip *gc, void *data)`

Callback function to check gpio_chip

Description

Similar to bus_find_device. It returns a reference to a gpio_chip as determined by a user supplied **match** callback. The callback should return 0 if the device doesn't match and non-zero if it does. If the callback is non-zero, this function will return to the caller and not iterate over any more gpio_chips.

int **`gpiochip_irq_map`**(*struct* irq_domain *d, unsigned int irq, irq_hw_number_t hwirq)

maps an IRQ into a GPIO irqchip

Parameters

`struct irq_domain *d`

the irqdomain used by this irqchip

`unsigned int irq`

the global irq number used by this GPIO irqchip irq

`irq_hw_number_t hwirq`

the local IRQ/GPIO line offset on this gpiochip

Description

This function will set up the mapping for a certain IRQ line on a gpiochip by assigning the gpiochip as chip data, and using the irqchip stored inside the gpiochip.

int **`gpiochip_irq_domain_activate`**(*struct* irq_domain *domain, *struct* irq_data *data, bool reserve)

Lock a GPIO to be used as an IRQ

Parameters

`struct irq_domain *domain`

The IRQ domain used by this IRQ chip

`struct irq_data *data`

Outermost irq_data associated with the IRQ `bool reserve`

If set, only reserve an interrupt vector instead of assigning one

Description

This function is a wrapper that calls `gpiochip_lock_as_irq()` and is to be used as the activate function for the `struct irq_domain_ops`. The host_data for the IRQ domain must be the `struct gpio_chip`.

---

void **`gpiochip_irq_domain_deactivate`**(*struct* irq_domain *domain, *struct* irq_data *data)

Unlock a GPIO used as an IRQ

Parameters

`struct irq_domain *domain`

The IRQ domain used by this IRQ chip

`struct irq_data *data`

Outermost irq_data associated with the IRQ

Description

This function is a wrapper that will call `gpiochip_unlock_as_irq()` and is to be used as the deactivate function for the `struct irq_domain_ops`. The host_data for the IRQ domain must be the `struct gpio_chip`.

---

int **`gpiochip_irqchip_add_domain`**(*struct* gpio_chip *gc, *struct* irq_domain *domain)

adds an irqdomain to a gpiochip

Parameters

`struct gpio_chip *gc`

the gpiochip to add the irqchip to

`struct irq_domain *domain`

the irqdomain to add to the gpiochip

Description

This function adds an IRQ domain to the gpiochip.

---

int `gpiochip_generic_request`(*struct* gpio_chip *gc, unsigned int offset)

request the gpio function for a pin

Parameters

`struct gpio_chip *gc`

the gpiochip owning the GPIO

`unsigned int offset`

the offset of the GPIO to request for GPIO function

---

void `gpiochip_generic_free`(*struct* gpio_chip *gc, unsigned int offset)

free the gpio function from a pin

Parameters

`struct gpio_chip *gc`

the gpiochip to request the gpio function for

`unsigned int offset`

the offset of the GPIO to free from GPIO function

---

int `gpiochip_generic_config`(*struct* gpio_chip *gc, unsigned int offset, unsigned long config)

apply configuration for a pin

Parameters

`struct gpio_chip *gc`

the gpiochip owning the GPIO

`unsigned int offset`

the offset of the GPIO to apply the configuration

`unsigned long config`

the configuration to be applied

---

int `gpiochip_add_pingroup_range`(*struct* gpio_chip *gc, *struct* pinctrl_dev *pctldev, unsigned int gpio_offset, *const* char *pin_group)

add a range for GPIO <-> pin mapping

Parameters

`struct gpio_chip *gc`

the gpiochip to add the range for

`struct pinctrl_dev *pctldev`

the pin controller to map to

`unsigned int gpio_offset`

the start offset in the current gpio_chip number space

`const char *pin_group`

name of the pin group inside the pin controller

Description

Calling this function directly from a DeviceTree-supported pinctrl driver is DEPRECATED. Please see Section 2.1 of Documentation/devicetree/bindings/gpio/gpio.txt on how to bind pinctrl and gpio drivers via the “gpio-ranges” property.

int `gpiochip_add_pin_range`(*struct* gpio_chip *gc, *const* char *pinctl_name, unsigned int gpio_offset, unsigned int pin_offset, unsigned int npins)

add a range for GPIO <-> pin mapping

Parameters

`struct gpio_chip *gc`

the gpiochip to add the range for

`const char *pinctl_name`

the dev_name() of the pin controller to map to

`unsigned int gpio_offset`

the start offset in the current gpio_chip number space

`unsigned int pin_offset`

the start offset in the pin controller number space

`unsigned int npins`

the number of pins from the offset of each pin space (GPIO and pin controller) to accumulate in this range

Return

0 on success, or a negative error-code on failure.

Description

Calling this function directly from a DeviceTree-supported pinctrl driver is DEPRECATED. Please see Section 2.1 of Documentation/devicetree/bindings/gpio/gpio.txt on how to bind pinctrl and gpio drivers via the "gpio-ranges" property.

---

void **`gpiochip_remove_pin_ranges`**(*struct* gpio_chip *gc)

remove all the GPIO <-> pin mappings

Parameters

`struct gpio_chip *gc`

the chip to remove all the mappings for

---

const char ***`gpiochip_is_requested`**(*struct* gpio_chip *gc, unsigned int offset)

return string iff signal was requested

Parameters

`struct gpio_chip *gc`

controller managing the signal

`unsigned int offset`

of signal within controller's 0..(ngpio - 1) range

Description

Returns NULL if the GPIO is not currently requested, else a string. The string returned is the label passed to gpio_request(); if none has been passed it is a meaningless, non-NULL constant.

This function is for use by GPIO controller drivers. The label can help with diagnostics, and knowing that the signal is used as a GPIO can help avoid accidentally multiplexing it to another controller.

---

struct gpio_desc ***`gpiochip_request_own_desc`**(*struct* gpio_chip *gc, unsigned int hwnum, *const* char *label, *enum* gpio_lookup_flags lflags, *enum* gpiod_flags dflags)

Allow GPIO chip to request its own descriptor

Parameters

`struct gpio_chip *gc`

GPIO chip

`unsigned int hwnum`

hardware number of the GPIO for which to request the descriptor

`const char *label`

label for the GPIO

`enum gpio_lookup_flags lflags`

lookup flags for this GPIO or 0 if default, this can be used to specify things like line inversion semantics with the machine flags such as GPIO_OUT_LOW

`enum gpiod_flags dflags`

descriptor request flags for this GPIO or 0 if default, this can be used to specify consumer semantics such as open drain

Description

Function allows GPIO chip drivers to request and use their own GPIO descriptors via gpiolib API. Difference to gpiod_request() is that this function will not increase reference count of the GPIO chip module. This allows the GPIO chip module to be unloaded as needed (we assume that the GPIO chip driver handles freeing the GPIOs it has requested).

Return

A pointer to the GPIO descriptor, or an ERR_PTR()-encoded negative error code on failure.

void **`gpiochip_free_own_desc`**(*struct* gpio_desc *desc)

Free GPIO requested by the chip driver

Parameters

`struct gpio_desc *desc`

GPIO descriptor to free

Description

Function frees the given GPIO requested previously with `gpiochip_request_own_desc()`.

int **`gpiod_direction_input`**(*struct* gpio_desc *desc)

set the GPIO direction to input

Parameters

`struct gpio_desc *desc`

GPIO to set to input

Description

Set the direction of the passed GPIO to input, such as `gpiod_get_value()` can be called safely on it.

Return 0 in case of success, else an error code.

---

int **`gpiod_direction_output_raw`**(*struct* gpio_desc *desc, int value)

set the GPIO direction to output

Parameters

`struct gpio_desc *desc`

GPIO to set to output

`int value`

initial output value of the GPIO

Description

Set the direction of the passed GPIO to output, such as `gpiod_set_value()` can be called safely on it. The initial value of the output must be specified as raw value on the physical line without regard for the ACTIVE_LOW status.

Return 0 in case of success, else an error code.

---

int **`gpiod_direction_output`**(*struct* gpio_desc *desc, int value)

set the GPIO direction to output

Parameters

`struct gpio_desc *desc`

GPIO to set to output

`int value`

initial output value of the GPIO

Description

Set the direction of the passed GPIO to output, such as `gpiod_set_value()` can be called safely on it. The initial value of the output must be specified as the logical value of the GPIO, i.e. taking its ACTIVE_LOW status into account.

Return 0 in case of success, else an error code.

---

int `gpiod_enable_hw_timestamp_ns`(*struct* gpio_desc *desc, unsigned long flags)

Enable hardware timestamp in nanoseconds.

Parameters

`struct gpio_desc *desc`

GPIO to enable.

`unsigned long flags`

Flags related to GPIO edge.

Description

Return 0 in case of success, else negative error code.

---

int `gpiod_disable_hw_timestamp_ns`(*struct* gpio_desc *desc, unsigned long flags)

Disable hardware timestamp.

Parameters

`struct gpio_desc *desc`

GPIO to disable.

`unsigned long flags`

Flags related to GPIO edge, same value as used during enable call.

Description

Return 0 in case of success, else negative error code.

---

int `gpiod_set_config`(*struct* gpio_desc *desc, unsigned long config)

sets **config** for a GPIO

Parameters

`struct gpio_desc *desc`

descriptor of the GPIO for which to set the configuration

`unsigned long config`

Same packed config format as generic pinconf

Return

0 on success, `-ENOTSUPP` if the controller doesn't support setting the configuration.

int `gpiod_set_debounce`(*struct* gpio_desc *desc, unsigned int debounce)

sets **debounce** time for a GPIO

Parameters

`struct gpio_desc *desc`

descriptor of the GPIO for which to set debounce time

`unsigned int debounce`

debounce time in microseconds

Return

0 on success, `-ENOTSUPP` if the controller doesn't support setting the debounce time.

int `gpiod_set_transitory`(*struct* gpio_desc *desc, bool transitory)

Lose or retain GPIO state on suspend or reset

Parameters

`struct gpio_desc *desc`

descriptor of the GPIO for which to configure persistence

`bool transitory`

True to lose state on suspend or reset, false for persistence

Return

0 on success, otherwise a negative error code.

`int gpiod_is_active_low(const struct gpio_desc *desc)`

test whether a GPIO is active-low or not

Parameters

`const struct gpio_desc *desc`

the gpio descriptor to test

Description

Returns 1 if the GPIO is active-low, 0 otherwise.

`void gpiod_toggle_active_low(struct gpio_desc *desc)`

toggle whether a GPIO is active-low or not

Parameters

`struct gpio_desc *desc`

the gpio descriptor to change

`int gpiod_get_raw_value(const struct gpio_desc *desc)`

return a gpio's raw value

Parameters

`const struct gpio_desc *desc`

gpio whose value will be returned

Description

Return the GPIO's raw value, i.e. the value of the physical line disregarding its ACTIVE_LOW status, or negative errno on failure.

This function can be called from contexts where we cannot sleep, and will complain if the GPIO chip functions potentially sleep.

`int gpiod_get_value(const struct gpio_desc *desc)`

return a gpio's value

Parameters

`const struct gpio_desc *desc`

gpio whose value will be returned

Description

Return the GPIO’s logical value, i.e. taking the ACTIVE_LOW status into account, or negative errno on failure.

This function can be called from contexts where we cannot sleep, and will complain if the GPIO chip functions potentially sleep.

---

int `gpiod_get_raw_array_value`(unsigned int array_size, *struct* gpio_desc **desc_array, *struct* gpio_array *array_info, unsigned long *value_bitmap)

read raw values from an array of GPIOs

Parameters

`unsigned int array_size`

number of elements in the descriptor array / value bitmap

`struct gpio_desc **desc_array`

array of GPIO descriptors whose values will be read

`struct gpio_array *array_info`

information on applicability of fast bitmap processing path

`unsigned long *value_bitmap`

bitmap to store the read values

Description

Read the raw values of the GPIOs, i.e. the values of the physical lines without regard for their ACTIVE_LOW status. Return 0 in case of success, else an error code.

This function can be called from contexts where we cannot sleep, and it will complain if the GPIO chip functions potentially sleep.

---

int `gpiod_get_array_value`(unsigned int array_size, *struct* gpio_desc **desc_array, *struct* gpio_array *array_info, unsigned long *value_bitmap)

read values from an array of GPIOs

Parameters

`unsigned int array_size`

number of elements in the descriptor array / value bitmap

`struct gpio_desc **desc_array`

array of GPIO descriptors whose values will be read

`struct gpio_array *array_info`

information on applicability of fast bitmap processing path

`unsigned long *value_bitmap`

bitmap to store the read values

Description

Read the logical values of the GPIOs, i.e. taking their ACTIVE_LOW status into account. Return 0 in case of success, else an error code.

This function can be called from contexts where we cannot sleep, and it will complain if the GPIO chip functions potentially sleep.

void **`gpiod_set_raw_value`**(*struct* gpio_desc *desc, int value)

assign a gpio's raw value

Parameters

`struct gpio_desc *desc`

gpio whose value will be assigned

`int value`

value to assign

Description

Set the raw value of the GPIO, i.e. the value of its physical line without regard for its ACTIVE_LOW status.

This function can be called from contexts where we cannot sleep, and will complain if the GPIO chip functions potentially sleep.

void **`gpiod_set_value`**(*struct* gpio_desc *desc, int value) assign a gpio's value

Parameters

`struct gpio_desc *desc`

gpio whose value will be assigned

`int value`

value to assign

Description

Set the logical value of the GPIO, i.e. taking its ACTIVE_LOW, OPEN_DRAIN and OPEN_SOURCE flags into account.

This function can be called from contexts where we cannot sleep, and will complain if the GPIO chip functions potentially sleep.

---

int `gpiod_set_raw_array_value`(unsigned int array_size, *struct* gpio_desc **desc_array, *struct* gpio_array *array_info, unsigned long *value_bitmap)

assign values to an array of GPIOs

Parameters

`unsigned int array_size`

number of elements in the descriptor array / value bitmap

`struct gpio_desc **desc_array`

array of GPIO descriptors whose values will be assigned

`struct gpio_array *array_info`

information on applicability of fast bitmap processing path

`unsigned long *value_bitmap`

bitmap of values to assign

Description

Set the raw values of the GPIOs, i.e. the values of the physical lines without regard for their ACTIVE_LOW status.

This function can be called from contexts where we cannot sleep, and will complain if the GPIO chip functions potentially sleep.

int **gpiod_set_array_value**(unsigned int array_size, *struct* gpio_desc **desc_array, *struct* gpio_array *array_info, unsigned long *value_bitmap)

assign values to an array of GPIOs

Parameters

`unsigned int array_size`

number of elements in the descriptor array / value bitmap

`struct gpio_desc **desc_array`

array of GPIO descriptors whose values will be assigned

`struct gpio_array *array_info`

information on applicability of fast bitmap processing path

`unsigned long *value_bitmap`

bitmap of values to assign

Description

Set the logical values of the GPIOs, i.e. taking their ACTIVE_LOW status into account.

This function can be called from contexts where we cannot sleep, and will complain if the GPIO chip functions potentially sleep.

int **gpiod_cansleep**(*const struct* gpio_desc *desc)

report whether gpio value access may sleep

Parameters

`const struct gpio_desc *desc`

gpio to check

int **gpiod_set_consumer_name**(*struct* gpio_desc *desc, *const* char *name)

set the consumer name for the descriptor

Parameters

`struct gpio_desc *desc`

gpio to set the consumer name on

`const char *name`

the new consumer name

---

int **`gpiod_to_irq`**(*const struct* gpio_desc *desc)

return the IRQ corresponding to a GPIO

Parameters

`const struct gpio_desc *desc`

gpio whose IRQ will be returned (already requested)

Description

Return the IRQ corresponding to the passed GPIO, or an error code in case of error.

---

int **`gpiochip_lock_as_irq`**(*struct* gpio_chip *gc, unsigned int offset)

lock a GPIO to be used as IRQ

Parameters

`struct gpio_chip *gc`

the chip the GPIO to lock belongs to

`unsigned int offset`

the offset of the GPIO to lock as IRQ

Description

This is used directly by GPIO drivers that want to lock down a certain GPIO line to be used for IRQs.

---

void **`gpiochip_unlock_as_irq`**(*struct* gpio_chip *gc, unsigned int offset)

unlock a GPIO used as IRQ

Parameters

`struct gpio_chip *gc`

the chip the GPIO to lock belongs to

`unsigned int offset`

the offset of the GPIO to lock as IRQ

Description

This is used directly by GPIO drivers that want to indicate that a certain GPIO is no longer used exclusively for IRQ.

---

int **`gpiod_get_raw_value_cansleep`**(*const struct* gpio_desc *desc)

return a gpio's raw value

Parameters

`const struct gpio_desc *desc`

gpio whose value will be returned

Description

Return the GPIO's raw value, i.e. the value of the physical line disregarding its ACTIVE_LOW status, or negative errno on failure.

This function is to be called from contexts that can sleep.

---

int **`gpiod_get_value_cansleep`**(*const struct* gpio_desc *desc)

return a gpio's value

Parameters

`const struct gpio_desc *desc`

gpio whose value will be returned

Description

Return the GPIO's logical value, i.e. taking the ACTIVE_LOW status into account, or negative errno on failure.

This function is to be called from contexts that can sleep.

---

int **`gpiod_get_raw_array_value_cansleep`**(unsigned int array_size, *struct* gpio_desc **desc_array, *struct* gpio_array *array_info, unsigned long *value_bitmap) read raw values from an array of GPIOs

Parameters

`unsigned int array_size`

number of elements in the descriptor array / value bitmap

`struct gpio_desc **desc_array`

array of GPIO descriptors whose values will be read

`struct gpio_array *array_info`

information on applicability of fast bitmap processing path

`unsigned long *value_bitmap`

bitmap to store the read values

Description

Read the raw values of the GPIOs, i.e. the values of the physical lines without regard for their ACTIVE_LOW status. Return 0 in case of success, else an error code.

This function is to be called from contexts that can sleep.

---

int `gpiod_get_array_value_cansleep`(unsigned int array_size, *struct* gpio_desc **desc_array, *struct* gpio_array *array_info, unsigned long *value_bitmap)

read values from an array of GPIOs

Parameters

`unsigned int array_size`

number of elements in the descriptor array / value bitmap

`struct gpio_desc **desc_array`

array of GPIO descriptors whose values will be read

`struct gpio_array *array_info`

information on applicability of fast bitmap processing path

`unsigned long *value_bitmap`

bitmap to store the read values

Description

Read the logical values of the GPIOs, i.e. taking their ACTIVE_LOW status into account. Return 0 in case of success, else an error code.

This function is to be called from contexts that can sleep.

void `gpiod_set_raw_value_cansleep`(*struct* gpio_desc *desc, int value)

assign a gpio’s raw value

Parameters

`struct gpio_desc *desc`

gpio whose value will be assigned

`int value`

value to assign

Description

Set the raw value of the GPIO, i.e. the value of its physical line without regard for its ACTIVE_LOW status.

This function is to be called from contexts that can sleep.

void `gpiod_set_value_cansleep`(*struct* gpio_desc *desc, int value)

assign a gpio’s value

Parameters

`struct gpio_desc *desc`

gpio whose value will be assigned

`int value`

value to assign

Description

Set the logical value of the GPIO, i.e. taking its ACTIVE_LOW status into account

This function is to be called from contexts that can sleep.

int `gpiod_set_raw_array_value_cansleep`(unsigned int array_size, *struct* gpio_desc **desc_array, *struct* gpio_array *array_info, unsigned long *value_bitmap)

assign values to an array of GPIOs

Parameters

`unsigned int array_size`

number of elements in the descriptor array / value bitmap

`struct gpio_desc **desc_array`

array of GPIO descriptors whose values will be assigned

`struct gpio_array *array_info`

information on applicability of fast bitmap processing path

`unsigned long *value_bitmap`

bitmap of values to assign

Description

Set the raw values of the GPIOs, i.e. the values of the physical lines without regard for their ACTIVE_LOW status.

This function is to be called from contexts that can sleep.

int `gpiod_set_array_value_cansleep`(unsigned int array_size, *struct* gpio_desc **desc_array, *struct* gpio_array *array_info, unsigned long *value_bitmap)

assign values to an array of GPIOs

Parameters

`unsigned int array_size`

number of elements in the descriptor array / value bitmap

`struct gpio_desc **desc_array`

array of GPIO descriptors whose values will be assigned

`struct gpio_array *array_info`

information on applicability of fast bitmap processing path

`unsigned long *value_bitmap`

bitmap of values to assign

Description

Set the logical values of the GPIOs, i.e. taking their ACTIVE_LOW status into account.

This function is to be called from contexts that can sleep.

---

void `gpiod_add_lookup_table`(*struct* gpiod_lookup_table *table)

register GPIO device consumers

Parameters

`struct gpiod_lookup_table *table`

table of consumers to register

---

void `gpiod_remove_lookup_table`(*struct* gpiod_lookup_table *table)

unregister GPIO device consumers

Parameters

`struct gpiod_lookup_table *table`

table of consumers to unregister

---

void `gpiod_add_hogs`(*struct* gpiod_hog *hogs)

register a set of GPIO hogs from machine code

Parameters

`struct gpiod_hog *hogs`

table of gpio hog entries with a zeroed sentinel at the end

***struct* gpio_desc *`fwnode_gpiod_get_index`(*struct* fwnode_handle *fwnode, *const* char *con_id, int index, *enum* gpiod_flags flags, *const* char *label)**

obtain a GPIO from firmware node

Parameters

`struct fwnode_handle *fwnode`

handle of the firmware node

`const char *con_id`

function within the GPIO consumer

`int index`

index of the GPIO to obtain for the consumer

`enum gpiod_flags flags`

GPIO initialization flags

`const char *label`

label to attach to the requested GPIO

Description

This function can be used for drivers that get their configuration from opaque firmware.

The function properly finds the corresponding GPIO using whatever is the underlying firmware interface and then makes sure that the GPIO descriptor is requested before it is returned to the caller.

In case of error an ERR_PTR() is returned.

Return

On successful request the GPIO pin is configured in accordance with provided **flags**.

---

int `gpiod_count`(*struct* device *dev, *const* char *con_id)

return the number of GPIOs associated with a device / function or -ENOENT if no GPIO has been assigned to the requested function

Parameters

`struct device *dev`

GPIO consumer, can be NULL for system-global GPIOs

`const char *con_id`

function within the GPIO consumer

---

struct gpio_desc *`gpiod_get`(*struct* device *dev, *const* char *con_id, *enum* gpiod_flags flags)

obtain a GPIO for a given GPIO function

Parameters

`struct device *dev`

GPIO consumer, can be NULL for system-global GPIOs `const char *con_id`

function within the GPIO consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

Return the GPIO descriptor corresponding to the function con_id of device dev, -ENOENT if no GPIO has been assigned to the requested function, or another IS_ERR() code if an error occurred while trying to acquire the GPIO.

***struct* gpio_desc *`gpiod_get_optional`(*struct* device *dev, *const* char *con_id, *enum* gpiod_flags flags)**

obtain an optional GPIO for a given GPIO function

Parameters

`struct device *dev`

GPIO consumer, can be NULL for system-global GPIOs

`const char *con_id`

function within the GPIO consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

This is equivalent to `gpiod_get()`, except that when no GPIO was assigned to the requested function it will return NULL. This is convenient for drivers that need to handle optional GPIOs.

***struct* gpio_desc *`gpiod_get_index`(*struct* device *dev, *const* char *con_id, unsigned int idx, *enum* gpiod_flags flags)**

obtain a GPIO from a multi-index GPIO function

Parameters

`struct device *dev`

GPIO consumer, can be NULL for system-global GPIOs

`const char *con_id`

function within the GPIO consumer

`unsigned int idx`

index of the GPIO to obtain in the consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

This variant of `gpiod_get()` allows to access GPIOs other than the first defined one for functions that define several GPIOs.

Return a valid GPIO descriptor, -ENOENT if no GPIO has been assigned to the requested function and/or index, or another IS_ERR() code if an error occurred while trying to acquire the GPIO.

***struct* gpio_desc *`fwnode_get_named_gpiod`(*struct* fwnode_handle *fwnode, *const* char *propname, int index, *enum* gpiod_flags dflags, *const* char *label)**

obtain a GPIO from firmware node

Parameters

`struct fwnode_handle *fwnode`

handle of the firmware node

`const char *propname`

name of the firmware property representing the GPIO

`int index`

index of the GPIO to obtain for the consumer

`enum gpiod_flags dflags`

GPIO initialization flags

`const char *label`

label to attach to the requested GPIO

Description

This function can be used for drivers that get their configuration from opaque firmware.

The function properly finds the corresponding GPIO using whatever is the underlying firmware interface and then makes sure that the GPIO descriptor is requested before it is returned to the caller.

In case of error an ERR_PTR() is returned.

Return

On successful request the GPIO pin is configured in accordance with provided **dflags**.

---

struct gpio_desc ***gpiod_get_index_optional**(*struct* device *dev, *const* char *con_id, unsigned int index, *enum* gpiod_flags flags)

obtain an optional GPIO from a multi-index GPIO function

Parameters

`struct device *dev`

GPIO consumer, can be NULL for system-global GPIOs

`const char *con_id`

function within the GPIO consumer

`unsigned int index`

index of the GPIO to obtain in the consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

This is equivalent to `gpiod_get_index()`, except that when no GPIO with the specified index was assigned to the requested function it will return NULL. This is convenient for drivers that need to handle optional GPIOs.

---

struct gpio_descs ***gpiod_get_array**(*struct* device *dev, *const* char *con_id, *enum* gpiod_flags flags)

obtain multiple GPIOs from a multi-index GPIO function

Parameters

`struct device *dev`

GPIO consumer, can be NULL for system-global GPIOs

`const char *con_id`

function within the GPIO consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

This function acquires all the GPIOs defined under a given function.

Return a struct gpio_descs containing an array of descriptors, -ENOENT if no GPIO has been assigned to the requested function, or another IS_ERR() code if an error occurred while trying to acquire the GPIOs.

---

struct gpio_descs ***gpiod_get_array_optional**(*struct* device *dev, *const* char *con_id, *enum* gpiod_flags flags)

obtain multiple GPIOs from a multi-index GPIO function

Parameters

`struct device *dev`

GPIO consumer, can be NULL for system-global GPIOs

`const char *con_id`

function within the GPIO consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

This is equivalent to `gpiod_get_array()`, except that when no GPIO was assigned to the requested function it will return NULL.

---

void **gpiod_put**(*struct* gpio_desc *desc)

dispose of a GPIO descriptor

Parameters

`struct gpio_desc *desc`

GPIO descriptor to dispose of

Description

No descriptor can be used after `gpiod_put()` has been called on it.

void `gpiod_put_array`(*struct* gpio_descs *descs)

dispose of multiple GPIO descriptors

Parameters

`struct gpio_descs *descs`

struct gpio_descs containing an array of descriptors

ACPI support

struct gpio_desc *`acpi_get_and_request_gpiod`(char *path, unsigned int pin, char *label)

Translate ACPI GPIO pin to GPIO descriptor and hold a refcount to the GPIO device.

Parameters

`char *path`

ACPI GPIO controller full path name, (e.g. “_SB.GPO1”)

`unsigned int pin`

ACPI GPIO pin number (0-based, controller-relative)

`char *label`

Label to pass to gpiod_request()

Description

This function is a simple pass-through to acpi_get_gpiod(), except that as it is intended for use outside of the GPIO layer (in a similar fashion to `gpiod_get_index()` for example) it also holds a reference to the GPIO device.

bool `acpi_gpio_get_io_resource`(*struct* acpi_resource *ares, *struct* acpi_resource_gpio **agpio)

Fetch details of an ACPI resource if it is a GPIO I/O resource or return False if not.

Parameters

`struct acpi_resource *ares`

Pointer to the ACPI resource to fetch

`struct acpi_resource_gpio **agpio`

Pointer to a `struct acpi_resource_gpio` to store the output pointer

void `acpi_gpiochip_request_interrupts`(*struct* gpio_chip *chip)

Register isr for gpio chip ACPI events

Parameters

`struct gpio_chip *chip`

GPIO chip

Description

ACPI5 platforms can use GPIO signaled ACPI events. These GPIO interrupts are handled by ACPI event methods which need to be called from the GPIO chip's interrupt handler. `acpi_gpiochip_request_interrupts()` finds out which GPIO pins have ACPI event methods and assigns interrupt handlers that calls the ACPI event methods for those pins.

void `acpi_gpiochip_free_interrupts`(*struct* gpio_chip *chip)

Free GPIO ACPI event interrupts.

Parameters

`struct gpio_chip *chip`

GPIO chip

Description

Free interrupts associated with GPIO ACPI event method for the given GPIO chip.

int `acpi_dev_gpio_irq_get_by`(*struct* acpi_device *adev, *const* char *name, int index)

Find GpioInt and translate it to Linux IRQ number

Parameters

`struct acpi_device *adev`

pointer to a ACPI device to get IRQ from

`const char *name`

optional name of GpioInt resource

`int index`

index of GpioInt resource (starting from `0`)

Description

If the device has one or more GpioInt resources, this function can be used to translate from the GPIO offset in the resource to the Linux IRQ number.

The function is idempotent, though each time it runs it will configure GPIO pin direction according to the flags in GpioInt resource.

The function takes optional **name** parameter. If the resource has a property name, then only those will be taken into account.

Return

Linux IRQ number (> `0`) on success, negative errno on failure.

Device tree support

struct gpio_desc *`gpiod_get_from_of_node`(*const struct* device_node *node, *const* char *propname, int index, *enum* gpiod_flags dflags, *const* char *label)

obtain a GPIO from an OF node

Parameters

`const struct device_node *node`

handle of the OF node

`const char *propname`

name of the DT property representing the GPIO

`int index`

index of the GPIO to obtain for the consumer

`enum gpiod_flags dflags`

GPIO initialization flags

`const char *label`

label to attach to the requested GPIO

Return

On successful request the GPIO pin is configured in accordance with provided **dflags**.

Description

In case of error an ERR_PTR() is returned.

int **`of_mm_gpiochip_add_data`**(*struct* device_node *np, *struct* of_mm_gpio_chip *mm_gc, void *data)

Add memory mapped GPIO chip (bank)

Parameters

`struct device_node *np`

device node of the GPIO chip

`struct of_mm_gpio_chip *mm_gc`

pointer to the of_mm_gpio_chip allocated structure

`void *data`

driver data to store in the `struct gpio_chip`

Description

To use this function you should allocate and fill mm_gc with:

1. In the gpio_chip structure: - all the callbacks - of_gpio_n_cells - of_xlate callback (optional)

3. In the of_mm_gpio_chip structure: - save_regs callback (optional)

If succeeded, this function will map bank's memory and will do all necessary work for you. Then you'll able to use .regs to manage GPIOs from the callbacks.

void **`of_mm_gpiochip_remove`**(*struct* of_mm_gpio_chip *mm_gc)

Remove memory mapped GPIO chip (bank)

Parameters

`struct of_mm_gpio_chip *mm_gc`

pointer to the of_mm_gpio_chip allocated structure

Device-managed API

struct gpio_desc ***`devm_gpiod_get`**(*struct* device *dev, *const* char *con_id, *enum* gpiod_flags flags)

Resource-managed `gpiod_get()`

Parameters

`struct device *dev`

GPIO consumer

`const char *con_id`

function within the GPIO consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

Managed `gpiod_get()`. GPIO descriptors returned from this function are automatically disposed on driver detach. See `gpiod_get()` for detailed information about behavior and return values.

---

struct gpio_desc ***devm_gpiod_get_optional**(*struct* device *dev, *const* char *con_id, *enum* gpiod_flags flags)

Resource-managed `gpiod_get_optional()`

Parameters

`struct device *dev`

GPIO consumer

`const char *con_id`

function within the GPIO consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

Managed `gpiod_get_optional()`. GPIO descriptors returned from this function are automatically disposed on driver detach. See `gpiod_get_optional()` for detailed information about behavior and return values.

---

struct gpio_desc ***devm_gpiod_get_index**(*struct* device *dev, *const* char *con_id, unsigned int idx, *enum* gpiod_flags flags)

Resource-managed `gpiod_get_index()`

Parameters

`struct device *dev`

GPIO consumer

`const char *con_id`

function within the GPIO consumer

`unsigned int idx`

index of the GPIO to obtain in the consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

Managed `gpiod_get_index()`. GPIO descriptors returned from this function are automatically disposed on driver detach. See `gpiod_get_index()` for detailed information about behavior and return values.

---

struct gpio_desc ***devm_gpiod_get_from_of_node**(*struct* device *dev, *const struct* device_node *node, *const* char *propname, int index, *enum* gpiod_flags dflags, *const* char *label)

obtain a GPIO from an OF node

Parameters

`struct device *dev`

device for lifecycle management

`const struct device_node *node`

handle of the OF node

`const char *propname`

name of the DT property representing the GPIO

`int index`

index of the GPIO to obtain for the consumer

`enum gpiod_flags dflags`

GPIO initialization flags

`const char *label`

label to attach to the requested GPIO

Return

On successful request the GPIO pin is configured in accordance with provided **dflags**.

Description

In case of error an ERR_PTR() is returned.

---

struct gpio_desc *`devm_fwnode_gpiod_get_index`(*struct* device *dev, *struct* fwnode_handle *fwnode, *const* char *con_id, int index, *enum* gpiod_flags flags, *const* char *label)

get a GPIO descriptor from a given node

Parameters

`struct device *dev`

GPIO consumer

`struct fwnode_handle *fwnode`

firmware node containing GPIO reference

`const char *con_id`

function within the GPIO consumer

`int index`

index of the GPIO to obtain in the consumer

`enum gpiod_flags flags`

GPIO initialization flags

`const char *label`

label to attach to the requested GPIO

Description

GPIO descriptors returned from this function are automatically disposed on driver detach.

On successful request the GPIO pin is configured in accordance with provided **flags**.

---

struct gpio_desc *`devm_gpiod_get_index_optional`(*struct* device *dev, *const* char *con_id, unsigned int index, *enum* gpiod_flags flags)

Resource-managed `gpiod_get_index_optional()`

Parameters

`struct device *dev`

GPIO consumer

`const char *con_id`

function within the GPIO consumer

`unsigned int index`

index of the GPIO to obtain in the consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

Managed `gpiod_get_index_optional()`. GPIO descriptors returned from this function are automatically disposed on driver detach. See `gpiod_get_index_optional()` for detailed information about behavior and return values.

---

struct **gpio_descs *`devm_gpiod_get_array`(*struct* device *dev, *const* char *con_id, *enum* gpiod_flags flags)**

Resource-managed `gpiod_get_array()`

Parameters

`struct device *dev`

GPIO consumer

`const char *con_id`

function within the GPIO consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

Managed `gpiod_get_array()`. GPIO descriptors returned from this function are automatically disposed on driver detach. See `gpiod_get_array()` for detailed information about behavior and return values.

---

struct **gpio_descs *`devm_gpiod_get_array_optional`(*struct* device *dev, *const* char *con_id, *enum* gpiod_flags flags)**

Resource-managed `gpiod_get_array_optional()`

Parameters

`struct device *dev`

GPIO consumer

`const char *con_id`

function within the GPIO consumer

`enum gpiod_flags flags`

optional GPIO initialization flags

Description

Managed `gpiod_get_array_optional()`. GPIO descriptors returned from this function are automatically disposed on driver detach. See `gpiod_get_array_optional()` for detailed information about behavior and return values.

void **devm_gpiod_put**(*struct* device *dev, *struct* gpio_desc *desc)

Resource-managed `gpiod_put()`

Parameters

`struct device *dev`

GPIO consumer

`struct gpio_desc *desc`

GPIO descriptor to dispose of

Description

Dispose of a GPIO descriptor obtained with `devm_gpiod_get()` or `devm_gpiod_get_index()`. Normally this function will not be called as the GPIO will be disposed of by the resource management code.

void **devm_gpiod_unhinge**(*struct* device *dev, *struct* gpio_desc *desc)

Remove resource management from a gpio descriptor

Parameters

`struct device *dev`

GPIO consumer

`struct gpio_desc *desc`

GPIO descriptor to remove resource management from

Description

Remove resource management from a GPIO descriptor. This is needed when you want to hand over lifecycle management of a descriptor to another mechanism.

void `devm_gpiod_put_array`(*struct* device *dev, *struct* gpio_descs *descs)

Resource-managed `gpiod_put_array()`

Parameters

`struct device *dev`

GPIO consumer

`struct gpio_descs *descs`

GPIO descriptor array to dispose of

Description

Dispose of an array of GPIO descriptors obtained with `devm_gpiod_get_array()`. Normally this function will not be called as the GPIOs will be disposed of by the resource management code.

int `devm_gpio_request`(*struct* device *dev, unsigned gpio, *const* char *label)

request a GPIO for a managed device

Parameters

`struct device *dev`

device to request the GPIO for

`unsigned gpio`

GPIO to allocate

`const char *label`

the name of the requested GPIO

Except for the extra **dev** argument, this function takes the same arguments and performs the same function as gpio_request(). GPIOs requested with this function will be automatically freed on driver detach.

int `devm_gpio_request_one`(*struct* device *dev, unsigned gpio, unsigned long flags, *const* char *label)

request a single GPIO with initial setup

Parameters

`struct device *dev`

device to request for

`unsigned gpio`

the GPIO number

`unsigned long flags`

GPIO configuration as specified by GPIOF_*

`const char *label`

a literal description string of this GPIO

int `devm_gpiochip_add_data_with_key`(*struct* device *dev, *struct* gpio_chip *gc, void *data, *struct* lock_class_key *lock_key, *struct* lock_class_key *request_key)

Resource managed gpiochip_add_data_with_key()

Parameters

`struct device *dev`

pointer to the device that gpio_chip belongs to.

`struct gpio_chip *gc`

the GPIO chip to register

`void *data`

driver-private data associated with this chip

`struct lock_class_key *lock_key`

lockdep class for IRQ lock

`struct lock_class_key *request_key`

lockdep class for IRQ request

Context

potentially before irqs will work

Description

The gpio chip automatically be released when the device is unbound.

Return

A negative errno if the chip can't be registered, such as because the gc->base is invalid or already associated with a different chip. Otherwise it returns zero as a success code.

sysfs helpers

int `gpiod_export`(*struct* gpio_desc *desc, bool direction_may_change)

export a GPIO through sysfs

Parameters

`struct gpio_desc *desc`

GPIO to make available, already requested

`bool direction_may_change`

true if userspace may change GPIO direction

Context

arch_initcall or later

Description

When drivers want to make a GPIO accessible to userspace after they have requested it – perhaps while debugging, or as part of their public interface – they may use this routine. If the GPIO can change direction (some can't) and the caller allows it, userspace will see "direction" sysfs attribute which may be used to change the gpio's direction. A "value" attribute will always be provided.

Returns zero on success, else an error.

int `gpiod_export_link`(*struct* device *dev, *const* char *name, *struct* gpio_desc *desc)

create a sysfs link to an exported GPIO node

Parameters

`struct device *dev`

device under which to create symlink

`const char *name`

name of the symlink

`struct gpio_desc *desc`

GPIO to create symlink to, already exported

Description

Set up a symlink from /sys/…/dev/name to /sys/class/gpio/gpioN node. Caller is responsible for unlinking.

Returns zero on success, else an error.

void `gpiod_unexport`(*struct* gpio_desc *desc)

reverse effect of `gpiod_export()`

Parameters

`struct gpio_desc *desc`

GPIO to make unavailable

Description

This is implicit on gpiod_free().