Parameters

txbuf	Pointer to a byte buffer which will be transmitted to the flash
rxbuf	Pointer to a byte buffer where data received from the flash will be written. txbuf and rxbuf may be the same buffer.
count	Length in bytes of txbuf and of rxbuf

◆ flash_get_unique_id()

```
void flash_get_unique_id ( uint8_t *  id_out )
```

Get flash unique 64 bit identifier.

Use a standard 4Bh RUID instruction to retrieve the 64 bit unique identifier from a flash device attached to the QSPI interface. Since there is a 1:1 association between the MCU and this flash, this also serves as a unique identifier for the board.

Parameters

id_out	Pointer to an 8-byte buffer to which the ID will be written

◆ flash_range_erase()

```
void flash_range_erase ( uint32_t  flash_offs,
                         size_t    count
                       )
```

Erase areas of flash.

Parameters

flash_offs	Offset into flash, in bytes, to start the erase. Must be aligned to a 4096-byte flash sector.
count	Number of bytes to be erased. Must be a multiple of 4096 bytes (one sector).

◆ flash_range_program()

```
void flash_range_program ( uint32_t          flash_offs,
                           const uint8_t *   data,
                           size_t            count
                         )
```

Program flash.

Parameters

flash_offs	Flash address of the first byte to be programmed. Must be aligned to a 256-byte flash page.
data	Pointer to the data to program into flash
count	Number of bytes to program. Must be a multiple of 256 bytes (one page).

hardware_gpio

Part of: Hardware APIs

Typedefs

- `typedef void(* gpio_irq_callback_t) (uint gpio, uint32_t event_mask)`

Enumerations

- **enum gpio_function {**
  GPIO_FUNC_XIP = 0 , GPIO_FUNC_SPI = 1 , GPIO_FUNC_UART = 2 , GPIO_FUNC_I2C = 3 ,
  GPIO_FUNC_PWM = 4 , GPIO_FUNC_SIO = 5 , GPIO_FUNC_PIO0 = 6 , GPIO_FUNC_PIO1 = 7 ,
  GPIO_FUNC_GPCK = 8 , GPIO_FUNC_USB = 9 , GPIO_FUNC_NULL = 0x1f
  }

  GPIO function definitions for use with function select. More...

- **enum gpio_irq_level { GPIO_IRQ_LEVEL_LOW = 0x1u , GPIO_IRQ_LEVEL_HIGH = 0x2u , GPIO_IRQ_EDGE_FALL = 0x4u , GPIO_IRQ_EDGE_RISE = 0x8u }**

  GPIO Interrupt level definitions (GPIO events) More...

- **enum gpio_slew_rate { GPIO_SLEW_RATE_SLOW = 0 , GPIO_SLEW_RATE_FAST = 1 }**

  Slew rate limiting levels for GPIO outputs. More...

- **enum gpio_drive_strength { GPIO_DRIVE_STRENGTH_2MA = 0 , GPIO_DRIVE_STRENGTH_4MA = 1 , GPIO_DRIVE_STRENGTH_8MA = 2 , GPIO_DRIVE_STRENGTH_12MA = 3 }**

  Drive strength levels for GPIO outputs. More...

Functions

- **void gpio_set_function (uint gpio, enum gpio_function fn)**

  Select GPIO function.

- **enum gpio_function gpio_get_function (uint gpio)**

  Determine current GPIO function.

- **void gpio_set_pulls (uint gpio, bool up, bool down)**

  Select up and down pulls on specific GPIO.

- **static void gpio_pull_up (uint gpio)**

  Set specified GPIO to be pulled up.

- **static bool gpio_is_pulled_up (uint gpio)**

  Determine if the specified GPIO is pulled up.

- **static void gpio_pull_down (uint gpio)**

  Set specified GPIO to be pulled down.

- **static bool gpio_is_pulled_down (uint gpio)**

  Determine if the specified GPIO is pulled down.

- **static void gpio_disable_pulls (uint gpio)**

  Disable pulls on specified GPIO.

- **void gpio_set_irqover (uint gpio, uint value)**

  Set GPIO IRQ override.

- **void gpio_set_outover (uint gpio, uint value)**

  Set GPIO output override.

- **void gpio_set_inover (uint gpio, uint value)**

  Select GPIO input override.

- **void gpio_set_oeover (uint gpio, uint value)**

  Select GPIO output enable override.

- **void gpio_set_input_enabled (uint gpio, bool enabled)**

  Enable GPIO input.

- **void gpio_set_input_hysteresis_enabled (uint gpio, bool enabled)**

  Enable/disable GPIO input hysteresis (Schmitt trigger)

- **bool gpio_is_input_hysteresis_enabled (uint gpio)**

  Determine whether input hysteresis is enabled on a specified GPIO.

- **void gpio_set_slew_rate (uint gpio, enum gpio_slew_rate slew)**

  Set slew rate for a specified GPIO.

- **enum gpio_slew_rate gpio_get_slew_rate (uint gpio)**

  Determine current slew rate for a specified GPIO.

- **void gpio_set_drive_strength (uint gpio, enum gpio_drive_strength drive)**

  Set drive strength for a specified GPIO.

- **enum gpio_drive_strength gpio_get_drive_strength (uint gpio)**

  Determine current slew rate for a specified GPIO.

- **void gpio_set_irq_enabled (uint gpio, uint32_t event_mask, bool enabled)**

  Enable or disable specific interrupt events for specified GPIO.

- **void gpio_set_irq_callback (gpio_irq_callback_t callback)**

  Set the generic callback used for GPIO IRQ events for the current core.

- **void gpio_set_irq_enabled_with_callback (uint gpio, uint32_t event_mask, bool enabled, gpio_irq_callback_t callback)**

  Convenience function which performs multiple GPIO IRQ related initializations.

- **void gpio_set_dormant_irq_enabled (uint gpio, uint32_t event_mask, bool enabled)**

  Enable dormant wake up interrupt for specified GPIO and events.

- **static uint32_t gpio_get_irq_event_mask (uint gpio)**

  Return the current interrupt status (pending events) for the given GPIO.

- **void gpio_acknowledge_irq (uint gpio, uint32_t event_mask)**

  Acknowledge a GPIO interrupt for the specified events on the calling core.

- **void gpio_add_raw_irq_handler_with_order_priority_masked (uint gpio_mask, irq_handler_t handler, uint8_t order_priority)**

  Adds a raw GPIO IRQ handler for the specified GPIOs on the current core.

- **static void gpio_add_raw_irq_handler_with_order_priority (uint gpio, irq_handler_t handler, uint8_t order_priority)**

Adds a raw GPIO IRQ handler for a specific GPIO on the current core.

- `void gpio_add_raw_irq_handler_masked (uint gpio_mask, irq_handler_t handler)`

  Adds a raw GPIO IRQ handler for the specified GPIOs on the current core.

- `static void gpio_add_raw_irq_handler (uint gpio, irq_handler_t handler)`

  Adds a raw GPIO IRQ handler for a specific GPIO on the current core.

- `void gpio_remove_raw_irq_handler_masked (uint gpio_mask, irq_handler_t handler)`

  Removes a raw GPIO IRQ handler for the specified GPIOs on the current core.

- `static void gpio_remove_raw_irq_handler (uint gpio, irq_handler_t handler)`

  Removes a raw GPIO IRQ handler for the specified GPIO on the current core.

- `void gpio_init (uint gpio)`

  Initialise a GPIO for (enabled I/O and set func to GPIO_FUNC_SIO)

- `void gpio_deinit (uint gpio)`

  Resets a GPIO back to the NULL function, i.e. disables it.

- `void gpio_init_mask (uint gpio_mask)`

  Initialise multiple GPIOs (enabled I/O and set func to GPIO_FUNC_SIO)

- `static bool gpio_get (uint gpio)`

  Get state of a single specified GPIO.

- `static uint32_t gpio_get_all (void)`

  Get raw value of all GPIOs.

- `static void gpio_set_mask (uint32_t mask)`

  Drive high every GPIO appearing in mask.

- `static void gpio_clr_mask (uint32_t mask)`

  Drive low every GPIO appearing in mask.

- `static void gpio_xor_mask (uint32_t mask)`

  Toggle every GPIO appearing in mask.

- `static void gpio_put_masked (uint32_t mask, uint32_t value)`

  Drive GPIO high/low depending on parameters.

- `static void gpio_put_all (uint32_t value)`

  Drive all pins simultaneously.

- `static void gpio_put (uint gpio, bool value)`

  Drive a single GPIO high/low.

- `static bool gpio_get_out_level (uint gpio)`

  Determine whether a GPIO is currently driven high or low.

- `static void gpio_set_dir_out_masked (uint32_t mask)`

  Set a number of GPIOs to output.

- **static void gpio_set_dir_in_masked (uint32_t mask)**

  Set a number of GPIOs to input.

- **static void gpio_set_dir_masked (uint32_t mask, uint32_t value)**

  Set multiple GPIO directions.

- **static void gpio_set_dir_all_bits (uint32_t values)**

  Set direction of all pins simultaneously.

- **static void gpio_set_dir (uint gpio, bool out)**

  Set a single GPIO direction.

- **static bool gpio_is_dir_out (uint gpio)**

  Check if a specific GPIO direction is OUT.

- **static uint gpio_get_dir (uint gpio)**

  Get a specific GPIO direction.

Detailed Description

General Purpose Input/Output (GPIO) API

RP2040 has 36 multi-functional General Purpose Input / Output (GPIO) pins, divided into two banks. In a typical use case, the pins in the QSPI bank (QSPI_SS, QSPI_SCLK and QSPI_SD0 to QSPI_SD3) are used to execute code from an external flash device, leaving the User bank (GPIO0 to GPIO29) for the programmer to use. All GPIOs support digital input and output, but GPIO26 to GPIO29 can also be used as inputs to the chip's Analogue to Digital Converter (ADC). Each GPIO can be controlled directly by software running on the processors, or by a number of other functional blocks.

The function allocated to each GPIO is selected by calling the gpio_set_function function.

NOTE

Not all functions are available on all pins.

Each GPIO can have one function selected at a time. Likewise, each peripheral input (e.g. UART0 RX) should only be selected on one *GPIO* at a time. If the same peripheral input is connected to multiple GPIOs, the peripheral sees the logical OR of these GPIO inputs. Please refer to the datasheet for more information on GPIO function select.

Function Select Table

GPIO	F1	F2	F3	F4	F5	F6	F7	F8	F9
0	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1		USB OVCUR DET
1	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1		USB VBUS DET
2	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1		USB VBUS EN
3	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1		USB OVCUR DET
4	SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1		USB VBUS DET
5	SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1		USB VBUS EN
6	SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1		USB OVCUR DET
7	SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1		USB VBUS DET
8	SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1		USB VBUS EN
9	SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1		USB OVCUR DET
10	SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1		USB VBUS DET
11	SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1		USB VBUS EN
12	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1		USB OVCUR DET
13	SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1		USB VBUS DET

14	SPI1 SCK	UART0 CTS	I2C1 SDA	PWM7 A	SIO	PIO0	PIO1		USB VBUS EN
15	SPI1 TX	UART0 RTS	I2C1 SCL	PWM7 B	SIO	PIO0	PIO1		USB OVCUR DET
16	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1		USB VBUS DET
17	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1		USB VBUS EN
18	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1		USB OVCUR DET
19	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1		USB VBUS DET
20	SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	CLOCK GPIN0	USB VBUS EN
21	SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	CLOCK GPOUT0	USB OVCUR DET
22	SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	CLOCK GPIN1	USB VBUS DET
23	SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	CLOCK GPOUT1	USB VBUS EN
24	SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	CLOCK GPOUT2	USB OVCUR DET
25	SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	CLOCK GPOUT3	USB VBUS DET
26	SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1		USB VBUS EN
27	SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1		USB OVCUR DET
28	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1		USB VBUS DET
29	SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1		USB VBUS EN

Typedef Documentation

◆ gpio_irq_callback_t

typedef void(* gpio_irq_callback_t) (uint gpio, uint32_t event_mask)

Callback function type for GPIO events

Parameters

gpio	Which GPIO caused this interrupt
event_mask	Which events caused this interrupt. See gpio_irq_level for details.

See also gpio_set_irq_enabled_with_callback() gpio_set_irq_callback()

Enumeration Type Documentation

◆ gpio_drive_strength

enum gpio_drive_strength

Drive strength levels for GPIO outputs.

Drive strength levels for GPIO outputs.

See also gpio_set_drive_strength

Enumerator	
GPIO_DRIVE_STRENGTH_2MA	2 mA nominal drive strength
GPIO_DRIVE_STRENGTH_4MA	4 mA nominal drive strength
GPIO_DRIVE_STRENGTH_8MA	8 mA nominal drive strength
GPIO_DRIVE_STRENGTH_12MA	12 mA nominal drive strength

◆ gpio_function

enum gpio_function

GPIO function definitions for use with function select.

GPIO function selectors

Each GPIO can have one function selected at a time. Likewise, each peripheral input (e.g. UART0 RX) should only be selected on one GPIO at a time. If the same peripheral input is connected to multiple GPIOs, the peripheral sees the logical OR of these GPIO inputs.

Please refer to the datasheet for more information on GPIO function selection.

◆ gpio_irq_level

enum gpio_irq_level

GPIO Interrupt level definitions (GPIO events)

GPIO Interrupt levels

An interrupt can be generated for every GPIO pin in 4 scenarios:

- Level High: the GPIO pin is a logical 1
- Level Low: the GPIO pin is a logical 0
- Edge High: the GPIO has transitioned from a logical 0 to a logical 1
- Edge Low: the GPIO has transitioned from a logical 1 to a logical 0

The level interrupts are not latched. This means that if the pin is a logical 1 and the level high interrupt is active, it will become inactive as soon as the pin changes to a logical 0. The edge interrupts are stored in the INTR register and can be cleared by writing to the INTR register.

◆ gpio_slew_rate

enum gpio_slew_rate

Slew rate limiting levels for GPIO outputs.

Slew rate limiting increases the minimum rise/fall time when a GPIO output is lightly loaded, which can help to reduce electromagnetic emissions.

See also gpio_set_slew_rate

Enumerator	
GPIO_SLEW_RATE_SLOW	Slew rate limiting enabled.
GPIO_SLEW_RATE_FAST	Slew rate limiting disabled.

Function Documentation

◆ gpio_acknowledge_irq()

```
void gpio_acknowledge_irq ( uint       gpio,
                            uint32_t   event_mask
                          )
```

Acknowledge a GPIO interrupt for the specified events on the calling core.

NOTE

This may be called with a mask of any of valid bits specified in gpio_irq_level, however it has no effect on *level* sensitive interrupts which remain pending while the GPIO is at the specified level. When handling *level* sensitive interrupts, you should generally disable the interrupt (see gpio_set_irq_enabled) and then set it up again later once the GPIO level has changed (or to catch the opposite level).

Parameters

gpio	GPIO number

NOTE

For callbacks set with gpio_set_irq_enabled_with_callback, or gpio_set_irq_callback, this function is called automatically.

Parameters

event_mask	Bitmask of events to clear. See gpio_irq_level for details.

◆ gpio_add_raw_irq_handler()

static void gpio_add_raw_irq_handler (uint ***gpio,***
irq_handler_t ***handler***
) inline static

Adds a raw GPIO IRQ handler for a specific GPIO on the current core.

In addition to the default mechanism of a single GPIO IRQ event callback per core (see gpio_set_irq_callback), it is possible to add explicit GPIO IRQ handlers which are called independent of the default event callback.

This method adds such a callback, and disables the "default" callback for the specified GPIO.

NOTE

Multiple raw handlers should not be added for the same GPIO, and this method will assert if you attempt to.

A raw handler should check for whichever GPIOs and events it handles, and acknowledge them itself; it might look something like:

```
void my_irq_handler(void) {
    if (gpio_get_irq_event_mask(my_gpio_num) & my_gpio_event_mask) {
       gpio_acknowledge_irq(my_gpio_num, my_gpio_event_mask);
      // handle the IRQ
    }
}
```

Parameters

gpio	the GPIO number that will no longer be passed to the default callback for this core
handler	the handler to add to the list of GPIO IRQ handlers for this core

◆ gpio_add_raw_irq_handler_masked()

void gpio_add_raw_irq_handler_masked (uint ***gpio_mask,***
irq_handler_t ***handler***
)

Adds a raw GPIO IRQ handler for the specified GPIOs on the current core.

In addition to the default mechanism of a single GPIO IRQ event callback per core (see gpio_set_irq_callback), it is possible to add explicit GPIO IRQ handlers which are called independent of the default event callback.

This method adds such a callback, and disables the "default" callback for the specified GPIOs.

NOTE

Multiple raw handlers should not be added for the same GPIOs, and this method will assert if you attempt to.

A raw handler should check for whichever GPIOs and events it handles, and acknowledge them itself; it might look something like:

```
void my_irq_handler(void) {
    if (gpio_get_irq_event_mask(my_gpio_num) & my_gpio_event_mask) {
       gpio_acknowledge_irq(my_gpio_num, my_gpio_event_mask);
      // handle the IRQ
    }
    if (gpio_get_irq_event_mask(my_gpio_num2) & my_gpio_event_mask2) {
       gpio_acknowledge_irq(my_gpio_num2, my_gpio_event_mask2);
      // handle the IRQ
    }
}
```

Parameters

gpio_mask	a bit mask of the GPIO numbers that will no longer be passed to the default callback for this core
handler	the handler to add to the list of GPIO IRQ handlers for this core

◆ gpio_add_raw_irq_handler_with_order_priority()

```
static void gpio_add_raw_irq_handler_with_order_priority ( uint          gpio,
                                                           irq_handler_t handler,
                                                           uint8_t       order_priority
                                                         )                          inline   static
```

Adds a raw GPIO IRQ handler for a specific GPIO on the current core.

In addition to the default mechanism of a single GPIO IRQ event callback per core (see gpio_set_irq_callback), it is possible to add explicit GPIO IRQ handlers which are called independent of the default callback. The order relative to the default callback can be controlled via the order_priority parameter(the default callback has the priority GPIO_IRQ_CALLBACK_ORDER_PRIORITY which defaults to the lowest priority with the intention of it running last).

This method adds such a callback, and disables the "default" callback for the specified GPIO.

NOTE

Multiple raw handlers should not be added for the same GPIO, and this method will assert if you attempt to.

A raw handler should check for whichever GPIOs and events it handles, and acknowledge them itself; it might look something like:

```
void my_irq_handler(void) {
    if (gpio_get_irq_event_mask(my_gpio_num) & my_gpio_event_mask) {
       gpio_acknowledge_irq(my_gpio_num, my_gpio_event_mask);
      // handle the IRQ
    }
}
```

Parameters

gpio	the GPIO number that will no longer be passed to the default callback for this core
handler	the handler to add to the list of GPIO IRQ handlers for this core
order_priority	the priority order to determine the relative position of the handler in the list of GPIO IRQ handlers for this core.

◆ gpio_add_raw_irq_handler_with_order_priority_masked()

```
void gpio_add_raw_irq_handler_with_order_priority_masked ( uint          gpio_mask,
```

irq_handler_t ***handler*,**
uint8_t ***order_priority***
)

Adds a raw GPIO IRQ handler for the specified GPIOs on the current core.

In addition to the default mechanism of a single GPIO IRQ event callback per core (see gpio_set_irq_callback), it is possible to add explicit GPIO IRQ handlers which are called independent of the default callback. The order relative to the default callback can be controlled via the order_priority parameter (the default callback has the priority GPIO_IRQ_CALLBACK_ORDER_PRIORITY which defaults to the lowest priority with the intention of it running last).

This method adds such an explicit GPIO IRQ handler, and disables the "default" callback for the specified GPIOs.

NOTE

Multiple raw handlers should not be added for the same GPIOs, and this method will assert if you attempt to.

A raw handler should check for whichever GPIOs and events it handles, and acknowledge them itself; it might look something like:

```
void my_irq_handler(void) {
    if (gpio_get_irq_event_mask(my_gpio_num) & my_gpio_event_mask) {
       gpio_acknowledge_irq(my_gpio_num, my_gpio_event_mask);
      // handle the IRQ
    }
    if (gpio_get_irq_event_mask(my_gpio_num2) & my_gpio_event_mask2) {
       gpio_acknowledge_irq(my_gpio_num2, my_gpio_event_mask2);
      // handle the IRQ
    }
}
```

Parameters

gpio_mask	a bit mask of the GPIO numbers that will no longer be passed to the default callback for this core
handler	the handler to add to the list of GPIO IRQ handlers for this core
order_priority	the priority order to determine the relative position of the handler in the list of GPIO IRQ handlers for this core.

◆ gpio_clr_mask()

static void gpio_clr_mask (uint32_t ***mask***) inline static

Drive low every GPIO appearing in mask.

Parameters

mask	Bitmask of GPIO values to clear, as bits 0-29

◆ gpio_deinit()

void gpio_deinit (uint ***gpio***)

Resets a GPIO back to the NULL function, i.e. disables it.

Parameters

gpio	GPIO number

◆ gpio_disable_pulls()

static void gpio_disable_pulls (uint *gpio*) inline static

Disable pulls on specified GPIO.

Parameters

gpio	GPIO number

◆ gpio_get()

static bool gpio_get (uint *gpio*) inline static

Get state of a single specified GPIO.

Parameters

gpio	GPIO number

Returns

Current state of the GPIO. 0 for low, non-zero for high

◆ gpio_get_all()

static uint32_t gpio_get_all (void) inline static

Get raw value of all GPIOs.

Returns

Bitmask of raw GPIO values, as bits 0-29

◆ gpio_get_dir()

static uint gpio_get_dir (uint *gpio*) inline static

Get a specific GPIO direction.

Parameters

gpio	GPIO number

Returns

1 for out, 0 for in

◆ gpio_get_drive_strength()

enum gpio_drive_strength gpio_get_drive_strength (uint *gpio*)

Determine current slew rate for a specified GPIO.

See also gpio_set_drive_strength

Parameters

gpio GPIO number

Returns

Current drive strength of that GPIO

◆ gpio_get_function()

enum gpio_function gpio_get_function (uint ***gpio***)

Determine current GPIO function.

Parameters

gpio GPIO number

Returns

Which GPIO function is currently selected from list gpio_function

◆ gpio_get_irq_event_mask()

static uint32_t gpio_get_irq_event_mask (uint ***gpio***) inline static

Return the current interrupt status (pending events) for the given GPIO.

Parameters

gpio GPIO number

Returns

Bitmask of events that are currently pending for the GPIO. See gpio_irq_level for details.

See also gpio_acknowledge_irq

◆ gpio_get_out_level()

static bool gpio_get_out_level (uint ***gpio***) inline static

Determine whether a GPIO is currently driven high or low.

This function returns the high/low output level most recently assigned to a GPIO via gpio_put() or similar. This is the value that is presented outward to the IO muxing, *not* the input level back from the pad (which can be read using gpio_get()).

To avoid races, this function must not be used for read-modify-write sequences when driving GPIOs – instead functions like gpio_put() should be used to atomically update GPIOs. This accessor is intended for debug use only.

Parameters

gpio GPIO number

Returns

true if the GPIO output level is high, false if low.

◆ gpio_get_slew_rate()

enum gpio_slew_rate gpio_get_slew_rate (uint ***gpio***)

Determine current slew rate for a specified GPIO.

See also gpio_set_slew_rate

Parameters

gpio	GPIO number

Returns

Current slew rate of that GPIO

◆ gpio_init()

void gpio_init (uint ***gpio***)

Initialise a GPIO for (enabled I/O and set func to GPIO_FUNC_SIO)

Clear the output enable (i.e. set to input). Clear any output value.

Parameters

gpio	GPIO number

◆ gpio_init_mask()

void gpio_init_mask (uint ***gpio_mask***)

Initialise multiple GPIOs (enabled I/O and set func to GPIO_FUNC_SIO)

Clear the output enable (i.e. set to input). Clear any output value.

Parameters

gpio_mask	Mask with 1 bit per GPIO number to initialize

◆ gpio_is_dir_out()

static bool gpio_is_dir_out (uint ***gpio***) inline static

Check if a specific GPIO direction is OUT.

Parameters

gpio	GPIO number

Returns

true if the direction for the pin is OUT

◆ gpio_is_input_hysteresis_enabled()

bool gpio_is_input_hysteresis_enabled (uint ***gpio***)

Determine whether input hysteresis is enabled on a specified GPIO.

See also gpio_set_input_hysteresis_enabled

Parameters

gpio	GPIO number

◆ gpio_is_pulled_down()

static bool gpio_is_pulled_down (uint ***gpio***) inline static

Determine if the specified GPIO is pulled down.

Parameters

gpio	GPIO number

Returns

true if the GPIO is pulled down

◆ gpio_is_pulled_up()

static bool gpio_is_pulled_up (uint ***gpio***) inline static

Determine if the specified GPIO is pulled up.

Parameters

gpio	GPIO number

Returns

true if the GPIO is pulled up

◆ gpio_pull_down()

static void gpio_pull_down (uint ***gpio***) inline static

Set specified GPIO to be pulled down.

Parameters

gpio	GPIO number

◆ gpio_pull_up()

```
static void gpio_pull_up ( uint  gpio )                inline  static
```

Set specified GPIO to be pulled up.

Parameters

gpio	GPIO number

◆ gpio_put()

```
static void gpio_put ( uint  gpio,
                       bool  value
                     )                                 inline  static
```

Drive a single GPIO high/low.

Parameters

gpio	GPIO number
value	If false clear the GPIO, otherwise set it.

◆ gpio_put_all()

```
static void gpio_put_all ( uint32_t  value )           inline  static
```

Drive all pins simultaneously.

Parameters

value	Bitmask of GPIO values to change, as bits 0-29

◆ gpio_put_masked()

```
static void gpio_put_masked ( uint32_t  mask,
                              uint32_t  value
                            )                          inline  static
```

Drive GPIO high/low depending on parameters.

Parameters

mask	Bitmask of GPIO values to change, as bits 0-29
value	Value to set

For each 1 bit in `mask`, drive that pin to the value given by corresponding bit in `value`, leaving other pins unchanged. Since this uses the TOGL alias, it is concurrency-safe with e.g. an IRQ bashing different pins from the same core.

◆ gpio_remove_raw_irq_handler()

```
static void gpio_remove_raw_irq_handler ( uint           gpio,
                                          irq_handler_t  handler
                                        )
```
inline static

Removes a raw GPIO IRQ handler for the specified GPIO on the current core.

In addition to the default mechanism of a single GPIO IRQ event callback per core (see gpio_set_irq_callback), it is possible to add explicit GPIO IRQ handlers which are called independent of the default event callback.

This method removes such a callback, and enables the "default" callback for the specified GPIO.

Parameters

gpio	the GPIO number that will now be passed to the default callback for this core
handler	the handler to remove from the list of GPIO IRQ handlers for this core

◆ gpio_remove_raw_irq_handler_masked()

```
void gpio_remove_raw_irq_handler_masked ( uint           gpio_mask,
                                          irq_handler_t  handler
                                        )
```

Removes a raw GPIO IRQ handler for the specified GPIOs on the current core.

In addition to the default mechanism of a single GPIO IRQ event callback per core (see gpio_set_irq_callback), it is possible to add explicit GPIO IRQ handlers which are called independent of the default event callback.

This method removes such a callback, and enables the "default" callback for the specified GPIOs.

Parameters

gpio_mask	a bit mask of the GPIO numbers that will now be passed to the default callback for this core
handler	the handler to remove from the list of GPIO IRQ handlers for this core

◆ gpio_set_dir()

```
static void gpio_set_dir ( uint  gpio,
                           bool  out
                         )
```
inline static

Set a single GPIO direction.

Parameters

gpio	GPIO number
out	true for out, false for in

◆ gpio_set_dir_all_bits()

```
static void gpio_set_dir_all_bits ( uint32_t  values )
```
inline static

Set direction of all pins simultaneously.

Parameters

values	individual settings for each gpio; for GPIO N, bit N is 1 for out, 0 for in

◆ gpio_set_dir_in_masked()

```
static void gpio_set_dir_in_masked ( uint32_t mask )    inline  static
```

Set a number of GPIOs to input.

Parameters

mask	Bitmask of GPIO to set to input, as bits 0-29

◆ gpio_set_dir_masked()

```
static void gpio_set_dir_masked ( uint32_t mask,
                                  uint32_t value
                                )                       inline  static
```

Set multiple GPIO directions.

Parameters

mask	Bitmask of GPIO to set to input, as bits 0-29
value	Values to set

For each 1 bit in "mask", switch that pin to the direction given by corresponding bit in "value", leaving other pins unchanged. E.g. gpio_set_dir_masked(0x3, 0x2); -> set pin 0 to input, pin 1 to output, simultaneously.

◆ gpio_set_dir_out_masked()

```
static void gpio_set_dir_out_masked ( uint32_t mask )    inline  static
```

Set a number of GPIOs to output.

Switch all GPIOs in "mask" to output

Parameters

mask	Bitmask of GPIO to set to output, as bits 0-29

◆ gpio_set_dormant_irq_enabled()

```
void gpio_set_dormant_irq_enabled ( uint     gpio,
                                    uint32_t event_mask,
                                    bool     enabled
                                  )
```

Enable dormant wake up interrupt for specified GPIO and events.

This configures IRQs to restart the XOSC or ROSC when they are disabled in dormant mode

Parameters

gpio	GPIO number
event_mask	Which events will cause an interrupt. See gpio_irq_level for details.

enabled	Enable/disable flag

◆ gpio_set_drive_strength()

```
void gpio_set_drive_strength ( uint                     gpio,
                               enum gpio_drive_strength drive
                             )
```

Set drive strength for a specified GPIO.

See also gpio_get_drive_strength

Parameters

gpio	GPIO number
drive	GPIO output drive strength

◆ gpio_set_function()

```
void gpio_set_function ( uint               gpio,
                         enum gpio_function fn
                       )
```

Select GPIO function.

Parameters

gpio	GPIO number
fn	Which GPIO function select to use from list gpio_function

◆ gpio_set_inover()

```
void gpio_set_inover ( uint gpio,
                       uint value
                     )
```

Select GPIO input override.

Parameters

gpio	GPIO number
value	See gpio_override

◆ gpio_set_input_enabled()

```
void gpio_set_input_enabled ( uint gpio,
                              bool enabled
                            )
```

Enable GPIO input.

Parameters

gpio	GPIO number

enabled	true to enable input on specified GPIO

◆ gpio_set_input_hysteresis_enabled()

```
void gpio_set_input_hysteresis_enabled ( uint  gpio,
                                         bool  enabled
                                       )
```

Enable/disable GPIO input hysteresis (Schmitt trigger)

Enable or disable the Schmitt trigger hysteresis on a given GPIO. This is enabled on all GPIOs by default. Disabling input hysteresis can lead to inconsistent readings when the input signal has very long rise or fall times, but slightly reduces the GPIO's input delay.

See also gpio_is_input_hysteresis_enabled

Parameters

gpio	GPIO number
enabled	true to enable input hysteresis on specified GPIO

◆ gpio_set_irq_callback()

```
void gpio_set_irq_callback ( gpio_irq_callback_t  callback )
```

Set the generic callback used for GPIO IRQ events for the current core.

This function sets the callback used for all GPIO IRQs on the current core that are not explicitly hooked via gpio_add_raw_irq_handler or other gpio_add_raw_irq_handler_ functions.

This function is called with the GPIO number and event mask for each of the (not explicitly hooked) GPIOs that have events enabled and that are pending (see gpio_get_irq_event_mask).

NOTE

The IO IRQs are independent per-processor. This function affects the processor that calls the function.

Parameters

callback	default user function to call on GPIO irq. Note only one of these can be set per processor.

◆ gpio_set_irq_enabled()

```
void gpio_set_irq_enabled ( uint      gpio,
                            uint32_t  event_mask,
                            bool      enabled
                          )
```

Enable or disable specific interrupt events for specified GPIO.

This function sets which GPIO events cause a GPIO interrupt on the calling core. See gpio_set_irq_callback, gpio_set_irq_enabled_with_callback and gpio_add_raw_irq_handler to set up a GPIO interrupt handler to handle the events.

NOTE

The IO IRQs are independent per-processor. This configures the interrupt events for the processor that calls the function.

Parameters

gpio	GPIO number
event_mask	Which events will cause an interrupt
enabled	Enable or disable flag

Events is a bitmask of the following gpio_irq_level values:

bit | constant | interrupt -—|--— 0 | GPIO_IRQ_LEVEL_LOW | Continuously while level is low 1 | GPIO_IRQ_LEVEL_HIGH | Continuously while level is high 2 | GPIO_IRQ_EDGE_FALL | On each transition from high to low 3 | GPIO_IRQ_EDGE_RISE | On each transition from low to high

which are specified in gpio_irq_level

◆ gpio_set_irq_enabled_with_callback()

```
void gpio_set_irq_enabled_with_callback ( uint                gpio,
                                          uint32_t            event_mask,
                                          bool                enabled,
                                          gpio_irq_callback_t callback
                                        )
```

Convenience function which performs multiple GPIO IRQ related initializations.

This method is a slightly eclectic mix of initialization, that:

- Updates whether the specified events for the specified GPIO causes an interrupt on the calling core based on the enable flag.
- Sets the callback handler for the calling core to callback (or clears the handler if the callback is NULL).
- Enables GPIO IRQs on the current core if enabled is true.

This method is commonly used to perform a one time setup, and following that any additional IRQs/events are enabled via gpio_set_irq_enabled. All GPIOs/events added in this way on the same core share the same callback; for multiple independent handlers for different GPIOs you should use gpio_add_raw_irq_handler and related functions.

This method is equivalent to:

```
gpio_set_irq_enabled(gpio, event_mask, enabled);
gpio_set_irq_callback(callback);
if (enabled) irq_set_enabled(IO_IRQ_BANK0, true);
```

NOTE

The IO IRQs are independent per-processor. This method affects only the processor that calls the function.

Parameters

gpio	GPIO number
event_mask	Which events will cause an interrupt. See gpio_irq_level for details.
enabled	Enable or disable flag
callback	user function to call on GPIO irq. if NULL, the callback is removed

◆ gpio_set_irqover()

```
void gpio_set_irqover ( uint gpio,
                        uint value
                      )
```

Set GPIO IRQ override.

Optionally invert a GPIO IRQ signal, or drive it high or low

Parameters

gpio	GPIO number
value	See gpio_override

◆ gpio_set_mask()

```
static void gpio_set_mask ( uint32_t mask )        inline   static
```

Drive high every GPIO appearing in mask.

Parameters

mask	Bitmask of GPIO values to set, as bits 0-29

◆ gpio_set_oeover()

```
void gpio_set_oeover ( uint  gpio,
                       uint  value
                     )
```

Select GPIO output enable override.

Parameters

gpio	GPIO number
value	See gpio_override

◆ gpio_set_outover()

```
void gpio_set_outover ( uint  gpio,
                        uint  value
                      )
```

Set GPIO output override.

Parameters

gpio	GPIO number
value	See gpio_override

◆ gpio_set_pulls()

```
void gpio_set_pulls ( uint  gpio,
                      bool  up,
                      bool  down
                    )
```

Select up and down pulls on specific GPIO.

Parameters

gpio	GPIO number

up	If true set a pull up on the GPIO
down	If true set a pull down on the GPIO

NOTE

On the RP2040, setting both pulls enables a "bus keep" function, i.e. a weak pull to whatever is current high/low state of GPIO.

◆ gpio_set_slew_rate()

```
void gpio_set_slew_rate ( uint                    gpio,
                          enum gpio_slew_rate     slew
                        )
```

Set slew rate for a specified GPIO.

See also gpio_get_slew_rate

Parameters

gpio	GPIO number
slew	GPIO output slew rate

◆ gpio_xor_mask()

```
static void gpio_xor_mask ( uint32_t mask )          inline  static
```

Toggle every GPIO appearing in mask.

Parameters

mask	Bitmask of GPIO values to toggle, as bits 0-29

hardware_i2c

Part of: Hardware APIs

Functions

- `uint i2c_init (i2c_inst_t *i2c, uint baudrate)`

  Initialise the I2C HW block.

- `void i2c_deinit (i2c_inst_t *i2c)`

  Disable the I2C HW block.

- `uint i2c_set_baudrate (i2c_inst_t *i2c, uint baudrate)`

  Set I2C baudrate.

- `void i2c_set_slave_mode (i2c_inst_t *i2c, bool slave, uint8_t addr)`

  Set I2C port to slave mode.

- `static uint i2c_hw_index (i2c_inst_t *i2c)`

  Convert I2C instance to hardware instance number.