API @iiot2k/gpio

Raspberry Pi gpio library.

The @iiot2k/gpio driver uses the gpio character devices interface from the Linux operating system.
The driver is loaded on call with `require("@iiot2k/gpio");`
On 64bit OS the driver ***gpio_arm64.node*** is loaded.
On 32bit OS the driver ***gpio_arm32.node*** is loaded.
If driver is unloaded function `deinit_gpio` is called for all pins.

pin Parameter

The Raspberry Pi has gpio pins from GPIO2 to GPIO27.
Valid `pin` parameter values are 2..27.

mode Parameter

Each gpio pin can be assigned a function.
The following table shows the different modes with `mode` parameter.

Constant	State-0	State-1	function
GPIO_MODE_INPUT_NOPULL	ground	+3.3V	floating input
GPIO_MODE_INPUT_PULLDOWN	open/ground	+3.3V	pulldown resistor input
GPIO_MODE_INPUT_PULLUP	open/+3.3V	ground	pullup resistor input
GPIO_MODE_OUTPUT	ground	+3.3V	output
GPIO_MODE_OUTPUT_SOURCE	Hi-Z	+3.3V	output source
GPIO_MODE_OUTPUT_SINK	Hi-Z	ground	output sink
GPIO_MODE_PWM	ground	+3.3V	pwm output pulse
GPIO_MODE_PWM_REALTIME	ground	+3.3V	pwm output pulse in realtime
GPIO_MODE_COUNTER_NOPULL	ground	+3.3V	floating input counter
GPIO_MODE_COUNTER_PULLDOWN	open/ground	+3.3V	pulldown resistor input counter
GPIO_MODE_COUNTER_PULLUP	open/+3.3V	ground	pullup resistor input counter
GPIO_MODE_SENSOR	ground	Hi-Z	output sink

Floating input/output is used when the pin is connected to another pin.
Input voltages more than +3.3V can destroy the input.
Hi-Z refers to an output signal state in which the signal is not being driven.
Please do not init gpio pins with special functions (i2c, spi, uart..).

Initialization functions

> Initializes the gpio with the mode.

`init_gpio(pin, mode, setval)`
pin: `<integer>` 2..27
mode: `<integer>` GPIO_MODE_...
setval: `<integer>` or `<boolean>`
return: `<boolean>` `true` on ok, `false` on error.
The ***setval*** parameter sets the **debounce time** in us for inputs,
and can set to 0 for disable debounce.
For outputs ***setval*** parameter sets initial state (0/1) of `true/false`.
If pin set to watch, false is returned.

> Changes the mode of gpio.

`change_gpio(pin, mode, setval)`
pin: `<integer>` 2..27
mode: `<integer>` GPIO_MODE_...
setval: `<integer>` or `<boolean>`
return: `<boolean>` `true` on ok, `false` on error.
The pin must be initialized with **init_gpio()**.
The ***setval*** parameter sets the **debounce time** in us for inputs,
and can set to 0 for disable debounce.
For outputs ***setval*** parameter sets initial state (0/1) of `true/false`.
Only mode of inputs and outputs can be changed.
If pin set to watch, false is returned.

Watch functions

edge Parameter

The `edge` parameter sets the watch direction of inputs.

Constant	Value	Function
GPIO_EDGE_RISING	0	check for change from 0 to 1
GPIO_EDGE_FALLING	1	check for change from 1 to 0
GPIO_EDGE_BOTH	2	check for change from 0 to 1 or 1 to 0

> Watch the gpio input for changes.

`watch_gpio(pin, mode, debounce, edge, callback)`
pin: `<integer>` 2..27
mode: `<integer>` GPIO_MODE_INPUT...
debounce: `<integer>` 0..
edge: `<integer>` GPIO_EDGE_...
callback: `<function>`

- (state, edge)
  state `<integer>` 0/1,
  edge `<integer>`GPIO_EDGE_RISING or GPIO_EDGE_FALLING.

return: `<number> 0/1` actual input state, `undefined` on error.

The ***debounce*** parameter sets the **debounce time** in us for inputs,
and can set to 0 for disable debounce.
The ***callback*** function is called if input state changes.
If pin set to watch, `undefined` is returned.

> Deinitializes the gpio and releases all resources.

`deinit_gpio(pin)`
pin: `<integer>` 2..27
return: `<boolean> true` on ok, `false` on error.
Stops also pwm and counter engine.
If pin set to watch, watch engine ist stopped.

Input/Output functions

> Gets gpio state as boolean.

`get_gpio(pin)`
pin: `<integer>` 2..27
return: `<boolean> false/true`, `undefined` on error.
The pin must be initialized as input or output.

> Gets gpio state as number.

`get_gpio_num(pin)`
pin: `<integer>` 2..27
return: `<integer>` 0/1, `undefined` on error.
The pin must be initialized as input or output.

> Sets gpio state of output.

`set_gpio(pin, value)`
pin: `<integer>` 2..27
value: `<integer>` 0/1 or `<boolean> true/false`
return: `<boolean> true` on ok, `false` on error.
The pin must be initialized as output.

> Toggle gpio state of output.

`toggle_gpio(pin)`
pin: `<integer>` 2..27
return: `<boolean> true` on ok, `false` on error.
The pin must be initialized as output.

Pulse Wide Modulation (PWM) Functions

Pulse Wide Modulation pulse is generated on output.
PWM can be used for example to adjust the brightness of LEDs.
Because PWM is generated with software, the accuracy of dutycycle is accurate up to approximately 800Hz.
For better accuracy use mode **GPIO_MODE_PWM_REALTIME**.
In **GPIO_MODE_PWM_REALTIME** mode the CPU load is higher.
The on+off time is 1/frequency (e.g. 1/100Hz = 10ms).
Dutycycle means the % time for on.
For example dutycycle 75% on 100Hz is 7.5ms on and 2.5ms off time.
A dutycycle of 0% turns output off.
A dutycycle of 100% turns output on.
The pin must be initialized as pwm (GPIO_MODE_PWM or GPIO_MODE_PWM_REALTIME).

> Sets pwm frequency and dutycycle.

`set_pwm(pin, frequency_hz, dutycycle)`
pin: `<integer>` 2..27
frequency_hz: `<integer>` 2..45000 (Hz)
dutycycle: `<integer>` 0..100 (%)
return: `<boolean>` `true` on ok, `false` on error.
Starts pwm engine if not startet.

> Gets pwm frequency.

`get_pwm_frequency(pin)`
pin: `<integer>` 2..27
return: `<integer>` 2..45000 (Hz), `undefined` on error.

> Gets pwm dutycycle.

`get_pwm_dutycycle(pin)`
pin: `<integer>` 2..27
return: 0..100 (%), `undefined` on error.

High Speed Counter Functions

Implements high speed software counter.
Counts on each change of input state from 0 to 1.
The pin must be initialized as counter (GPIO_MODE_COUNTER_..)

cnt_mode Parameter

The `cnt_mode` parameter sets the count direction of counter.

Constant	Value	Function
C_UP	0	counts up to `counter_high`
C_DOWN	1	counts down to zero

> Sets counter hardware reset pin.

`set_counter_reset_pin(pin, pin_reset)`
pin: `<integer>` 2..27
pin_reset: `<integer>` 2..27
return: `<boolean> true` on ok, `false` on error.
Sets counter hardware reset pin.
`pin_reset` must init to input (GPIO_MODE_INPUT_..) before call `set_counter`.
If `pin_reset` changes input state from 0 to 1, the counter is reset.

> Sets counter hardware output pin.

`set_counter_output_pin(pin, pin_output)`
pin: `<integer>` 2..27
pin_output: `<integer>` 2..27
return: `<boolean> true` on ok, `false` on error.
Sets counter hardware output pin.
`pin_output` must init to output (GPIO_MODE_OUTPUT_..) before call `set_counter`.
`pin_output` state is equal call of `is_counter_onlimit`.

> Sets counter.

`set_counter(pin, counter_high, cnt_mode)`
pin: `<integer>` 2..27
counter_high: `<integer>` 1..4294967294
cnt_mode: `<integer>` **C_UP**, **C_DOWN**
return: `<boolean> true` on ok, `false` on error.
Starts counter engine if not startet.
Resets also counter (see reset_counter).

> Resets counter.

`reset_counter(pin)`
pin: `<integer>` 2..27
return: `<boolean> true` on ok, `false` on error.
cnt_mode = **C_UP**: counter is set to 0.
cnt_mode = **C_DOWN**: counter is set to `counter_high`.

> Gets counter value.

`get_counter(pin)`
pin: `<integer>` 2..27
return: `<integer>` counter value (0..4294967294), `undefined` on error

> Gets counter_high value.

`get_counter_high(pin)`
pin: `<integer>` 2..27
return: `<integer> counter_high` value (1..4294967294), `undefined` on error

> Gets counter mode.

`get_counter_mode(pin)`
pin: `<integer>` 2..27
return : `<integer>` **C_UP**, **C_DOWN**, `undefined` on error

> Check if counter is on limits.

`is_counter_onlimit(pin)`
pin: `<integer>` 2..27
return : `<boolean> true/false`, `undefined` on error
cnt_mode = **C_UP**: returns `true` when counter is equal ***counter_high***.
cnt_mode = **C_DOWN**: returns `true` when counter is equal 0.

Temperature Sensors with 1-wire protocol

The functions reads the DS18B20 temperature.
You can connect multiple sensors in parallel.
It is important that a 4.7k pullup resistor is connected.
The library does not support the parasite mode.
Please do not activate the 1-wire subsystem of the raspberry pi.
For high-performance reading, all sensors must have set to the same resolution.
The pin must be initialized as GPIO_MODE_SENSOR.

res Parameter

The `res` (resolution) parameter sets the resolution of ds18b20 sensors.

Constant	Value	Resolution	Conversion Time	Temp. Steps
RES_SENSOR_9	0	9 bit	94ms	0.5°
RES_SENSOR_10	1	10 bit	187ms	0.25°
RES_SENSOR_11	2	11 bit	375ms	0.125°
RES_SENSOR_12	3	12 bit	750ms	0.0625°

> Init all sensors to resolution.

`init_sensor(pin, res)`
pin: `<integer>` 2..27
res: `<integer>` RES_SENSOR_..
return: `<boolean> true` on ok, `false` on error.

> Scans all sensors.

`scan_sensor(pin, callback)`
pin: `<integer>` 2..27
callback: `<function>`
`(ret) <boolean> true` on ok, `false` on error.

`scan_sensor_sync(pin)`
pin: `<integer>` 2..27
return: `<boolean> true` on ok, `false` on error.

> List all sensors.

`list_sensor(pin)`
pin: `<integer>` 2..27
return: `<string array>` list of sensors id in format 28-HHHHHHHHHHHH (hex), `undefined` on error.

Get sensor count.

`get_sensor_count(pin)`
pin: `<integer>` 2..27
return: `<integer>` number of sensors, `undefined` on error.

Read all sensors.

`read_sensor(pin, fh, callback)`
pin: `<integer>` 2..27
fh: `<boolean>` `false:` output is celsius, `true:` output is fahrenheit.
callback: `<function>`
`(data) <number array>` temperature of sensors, `undefined` on error.

`read_sensor_sync(pin, fh)`
pin: `<integer>` 2..27
fh: `<boolean>` `false:` output is celsius, `true:` output is fahrenheit.
return: `<number array>` temperature of sensors, `undefined` on error.

Read one sensor.

`read_one_sensor_promise(pin, id, fh, callback)`
pin: `<integer>` 2..27
id: `<string>` id of sensor in format 28-HHHHHHHHHHHH (hex).
fh: `<boolean>` `false:` output is celsius, `true:` output is fahrenheit.
callback: `<function>`
`(data) <number>` temperature of sensor, `undefined` on error or no sensors.

`read_one_sensor_sync(pin, id, fh)`
pin: `<integer>` 2..27
id: `<string>` id of sensor in format 28-HHHHHHHHHHHH (hex).
fh: `<boolean>` `false:` output is celsius, `true:` output is fahrenheit.
return: `<number>` temperature of sensor, `undefined` on error or no sensors.