libregpio

Release 0.0.1

Roberto Chen

Feb 18, 2023

CONTENTS:

`libregpio` is a python module that aims to provide basic, straight-forward GPIO input/output operations for Libre Computer "Le Potato" using `gpiod`

Note: This is an enthusiast project. It is not an official Libre Computer's package.

CHAPTER

ONE

INSTALLATION

Before installation it is required to have `gpiod` installed on your board.

```
$sudo apt install gpiod
```

This module is not yet available to install via **pip**.

Source code: https://github.com/c0t088/libregpio

CHAPTER

TWO

IMPORTING THE MODULE

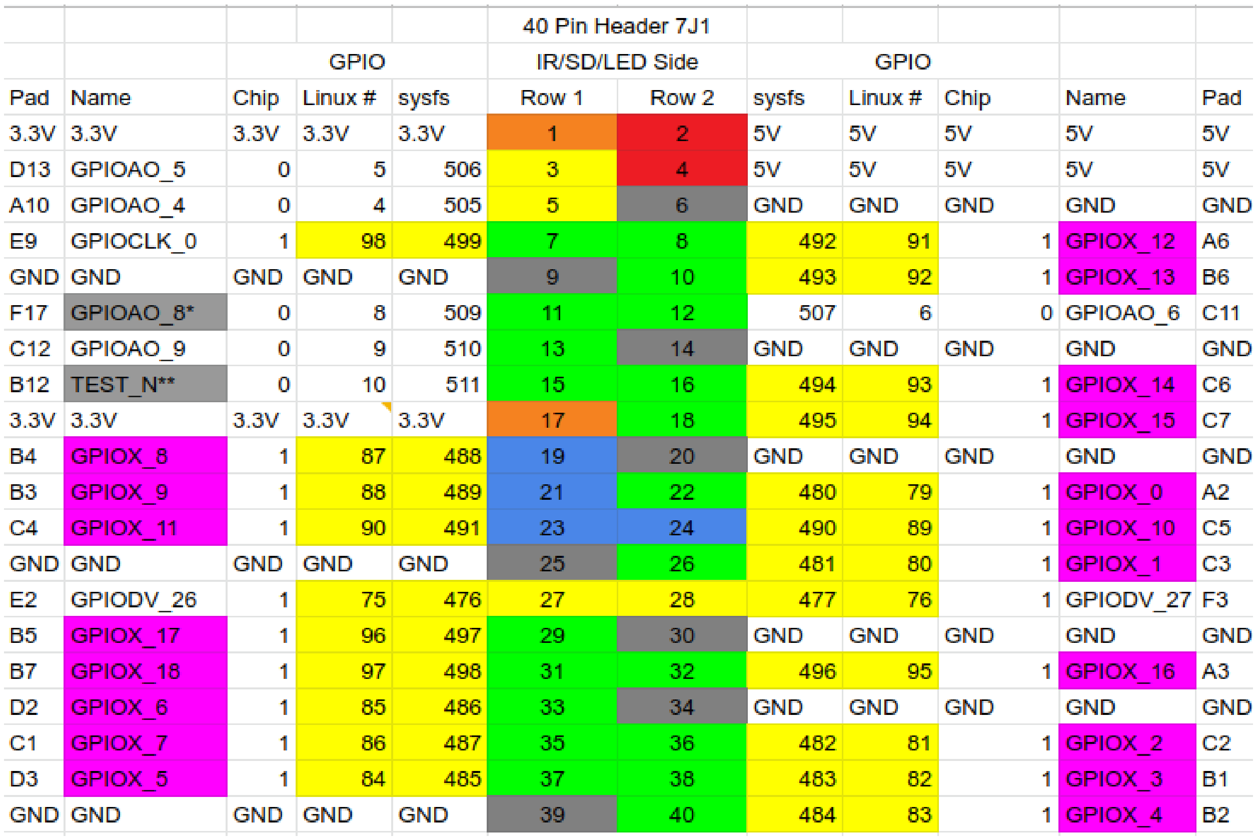
To import the libregpio module:

```
import libregpio as GPIO
```

This way allows you to refer to it as GPIO for the rest of your program.

PIN REFERENCE

This module is designed to work with the 40-pin chip of Libre Computer AML-S905X-CC “LePotato”.



					40 Pin Header 7J1						
			GPIO		IR/SD/LED Side			GPIO			
Pad	Name	Chip	Linux #	sysfs	Row 1	Row 2	sysfs	Linux #	Chip	Name	Pad
3.3V	3.3V	3.3V	3.3V	3.3V	1	2	5V	5V	5V	5V	5V
D13	GPIOAO_5	0	5	506	3	4	5V	5V	5V	5V	5V
A10	GPIOAO_4	0	4	505	5	6	GND	GND	GND	GND	GND
E9	GPIOCLK_0	1	98	499	7	8	492	91	1	GPIOX_12	A6
GND	GND	GND	GND	GND	9	10	493	92	1	GPIOX_13	B6
F17	GPIOAO_8*	0	8	509	11	12	507	6	0	GPIOAO_6	C11
C12	GPIOAO_9	0	9	510	13	14	GND	GND	GND	GND	GND
B12	TEST_N**	0	10	511	15	16	494	93	1	GPIOX_14	C6
3.3V	3.3V	3.3V	3.3V	3.3V	17	18	495	94	1	GPIOX_15	C7
B4	GPIOX_8	1	87	488	19	20	GND	GND	GND	GND	GND
B3	GPIOX_9	1	88	489	21	22	480	79	1	GPIOX_0	A2
C4	GPIOX_11	1	90	491	23	24	490	89	1	GPIOX_10	C5
GND	GND	GND	GND	GND	25	26	481	80	1	GPIOX_1	C3
E2	GPIODV_26	1	75	476	27	28	477	76	1	GPIODV_27	F3
B5	GPIOX_17	1	96	497	29	30	GND	GND	GND	GND	GND
B7	GPIOX_18	1	97	498	31	32	496	95	1	GPIOX_16	A3
D2	GPIOX_6	1	85	486	33	34	GND	GND	GND	GND	GND
C1	GPIOX_7	1	86	487	35	36	482	81	1	GPIOX_2	C2
D3	GPIOX_5	1	84	485	37	38	483	82	1	GPIOX_3	B1
GND	GND	GND	GND	GND	39	40	484	83	1	GPIOX_4	B2

Note: Please, see Libre Computer’s GPIO Headers Reference for full functions documentation: https://docs.google.com/spreadsheets/d/1U3z0Gb8HUEfCIMkvqzmhMpJfzRqjPXq7mFLC-hvbKlE/edit#gid=0

To access GPIO pins with this module, a class instance needs to be created. The pins are referred to by their GPIO name.

This is an example of an `IN` (input) class instance set to use ‘GPIOX_4’ pin:

```python
import libregpio as GPIO

 a_pin = GPIO.IN('GPIOX_4')
```

CHAPTER

FOUR

HOW TO USE

As noted in the previous section, GPIO pins are handled as class instances based on their intended use. Here we will run through some code examples.

Note: Please, take notice that the `cleanup()` method is used at the end of every example. This is recommended to avoid leaving any pins on a high state after the end of your program.

4.1 IN Class examples

This section contains examples on how to use GPIO pins as inputs.

4.1.1 Read a current GPIO value

In this example we create an instance of the `libregpio.IN` class and call the `input` method to read the pin value:

```python
import libregpio as GPIO

# set pin GPIOX_12 to be used as an input
pin = GPIO.IN('GPIOX_12')

# read pin value
value = pin.input()

# print read value
print(value)

GPIO.cleanup()
```

4.1.2 Pull up and Pull down resistors

When using a pin as an input it may be at a floating state, sending unreliable values. To prevent this, the `bias` parameter can be used in the `input` method to set `pull-up` or `pull-down` resistors.

This is the same example as above, but setting a pull-down bias:

```
import libregpio as GPIO

# set pin GPIOX_12 to be used as an input
pin = GPIO.IN('GPIOX_12')

# read pin value with a pull-down resistor
value = pin.input(bias='pull-down')

# print read value
print(value)

GPIO.cleanup()
```

4.1.3 Wait for an edge event

In some applications you may want your program to wait for a falling-edge or rising-edge event. For this, you can use the `wait_for_edge` method.

In this example we are using a PIR motion sensor connected to the GPIOX_12 pin. The program waits for a rising-edge event before printing the corresponding value:

```
import libregpio as GPIO

# set pin GPIOX_12 to be used as an input
pin = GPIO.IN('GPIOX_12')

# wait for a rising-edge event. Bias is set to pull-down
value = pin.wait_for_edge(bias='pull-down', edge='rising')

# print event value
print(value)

GPIO.cleanup()
```

Note: You can use the `num_events` parameter if you want to wait for more than one event occurrence.

4.2 OUT Class examples

In this section, we will turn an LED on for three seconds using the different methods of the `libregpio.OUT` class.

4.2.1 output method

```python
import libregpio as GPIO
from time import sleep

# set pin GPIOX_5 to be used as an output
led = GPIO.OUT('GPIOX_5')

# send a 1 value and return it to 0 after 3 seconds
led.output(1)
sleep(3)
led.output(0)

GPIO.cleanup()
```

4.2.2 high and low methods

```python
import libregpio as GPIO
from time import sleep

# set pin GPIOX_5 to be used as an output
led = GPIO.OUT('GPIOX_5')

# set the pin output to high and return to low after 3 seconds
led.high()
sleep(3)
led.low()

GPIO.cleanup()
```

4.2.3 toggle method

```python
import libregpio as GPIO
from time import sleep

# set pin GPIOX_5 to be used as an output
led = GPIO.OUT('GPIOX_5')

# set the pin output to high and return to low after 3 seconds
led.toggle()
sleep(3)
led.toggle()

GPIO.cleanup()
```

CHAPTER

FIVE

API DOCUMENTATION

Warning: Although this module contains a PWM class, it is not currently working properly. Be aware that using this class and its methods can lead to unexpected results.

`class libregpio.IN`(*pin*)

Bases: `object`

This is a class representantion of a GPIO pin to be used as an input.

Parameters
`pin` (*str*) – GPIO pin name (i.e. GPIOX_4)

`input`(*bias='as-is'*)

Read an input value from a libregpio.IN object.

This method can read the pin input value at a given time.

Use the bias parameter to enable pull-up or pull-down modes.

Parameters
`bias` (*str, optional*) – `pull-up`, `pull-down`, `as-is`, `disable`

Returns
Input value read from GPIO pin (i.e. `0` or 1)

Return type
int

`wait_for_edge`(*bias='as-is'*, *edge='rising'*, *num_events=1*, *active_low=False*)

Returns an input value when a specific edge event is detected. This method is designed to stop your program execution until an event is detected.

Parameters

- `bias` (*str, optional*) – `pull-up`, `pull-down`, `as-is`, `disable`
- `edge` (*str, optional*) – Type of event to wait for (`rising`, `falling`), defaults to 'rising'
- `num_events` (*Boolean, optional*) – number of events to wait for. defaults to 1
- `active_low` – Set pin to active-low state (`True`, `False`). defaults to False.

Returns
1 for rising `0` for falling

Return type
int

`class libregpio.OUT(`*pin*`)`

Bases: `object`

This is a class representantion of a GPIO pin to be used as an output.

Parameters
`pin` (*str*) – GPIO pin name (i.e. GPIOX_4)

`active_low()`

Set libregpio.OUT object to active_low.

`high()`

Set a value of 1 to a libregpio.OUT object.

`low()`

Set a value of `0` to a libregpio.OUT object

`output(`*value*`)`

Set an output value to a libregpio.OUT object (i.e. `0` or 1).

Parameters
`value` (*int*) – output value to be sent to GPIO pin

`toggle()`

Toggle output value of a GPIO pin

`class libregpio.PWM(`*pin*, *duty_cycle*, *freq*`)`

Bases: `Thread`

This is a class representantion of a GPIO pin to be used as an PWM output.

Use only with pins compatible with PWM (pulse width modulation).

Creating the class instance does not automatically sends a PWM output.

Parameters

- `pin` (*str*) – GPIO pin name (i.e. GPIOX_4)
- `duty_cycle` (*int*) – duty cycle percentage
- `freq` (*float*) – frequency in Hertz

`change_duty_cycle(`*duty_cycle*`)`

Modify the current duty cycle

Parameters
`duty_cycle` (*int*) – duty cycle percentage

`change_freq(`*freq*`)`

Modify the current frequency

Parameters
`freq` (*float*) – frequency in Hertz

`pulse_loop()`

This method is called by `start()` to loop the pulse output on a different thread

Do not call this method outside of this class.

`start`(*duty_cycle=None*)

Start the PWM output.

You can update the duty cycle when starting this method.

Parameters
`duty_cycle` (*`int, optional`*) – duty cycle percentage, defaults to None

`stop()`

Stop the PWM output

It 'cleans up' the GPIO pin.

`libregpio.`**`cleanup`**(*pins=None*)

By Default, it sets all pins to `0` but you can pass a list if only specific pins need to be cleaned up.

It is recommended to use this method at the end of your program.

Parameters
`pins` (*`iterable, optional`*) – list/tuple of pin or pins by name, defaults to `None`

`libregpio.`**`set_chip`**(*pin_name*)

Select the gpio chip corresponding to the pin. Do not call this function.

Parameters
`pin_name` (*`str`*) – gpio pin name

Returns
gpio chip

Return type
str

CHAPTER

SIX

COMPATIBILITY WITH OTHER BOARDS

This module is designed to work with Libre Computer's "LePotato". However, it can be mapped to different boards if needed, provided they are work with gpiod.

To achieve this, you need to modify the pin_mapping.py file to match your board.

```
# Modify this dictionary to your preffered pin names and corresponding
# linux number of said pins
PIN_NAME = {
"GPIOAO_5": 5,
"GPIOAO_4": 4,
"GPIOCLK_0": 98,
.
.
.
```

And you need to modify the set_chip() method in the libregpio.py file to set the corresponding chip of every pin.

```
def set_chip(pin_name):
# modify this code to match your board gpio chips
    chip_zero = ['GPIOAO_5','GPIOAO_4','GPIOAO_8','GPIOAO_9','TEST_N','GPIOAO_6']
    if pin_name in chip_zero:
        chip = 0
    else:
        chip = 1
    return str(chip)
```

CHAPTER

SEVEN

GITHUB

The source code is available to clone at: https://github.com/c0t088/libregpio

CHAPTER

EIGHT

REFERENCES

- OPi.GPIO (Copyright (c) 2018 Richard Hull): https://github.com/rm-hull/OPi.GPIO
- Libre Computer Header Reference: https://docs.google.com/spreadsheets/d/1U3z0Gb8HUEfCIMkvqzmhMpJfzRqjPXq7mFLC-hvbKlE/edit#gid=0

CHAPTER

NINE

CHANGELOG

Version	Description	Date
0.0.1	Initial Version	2022-11-06

CHAPTER

TEN

MIT LICENCE

PYTHON MODULE INDEX

INDEX