PiFace Common Documentation

Release 4.1.2

Thomas Preston

February 17, 2015

Contents

1 Installation **3**
- 1.1 Install . . . 3
- 1.2 Uninstall . . . 4

2 Example **5**
- 2.1 MCP23S17 . . . 5

3 Reference **7**
- 3.1 Core . . . 7
- 3.2 SPI . . . 7
- 3.3 Interrupts . . . 8
- 3.4 MCP23S17 . . . 10

4 Indices and tables **13**

Python Module Index **15**

The pifacecommon Python module is required for all PiFace extensions. It provides functions and classes for generic interfacing with the boards over SPI.

Links:

- Blog
- GitHub
- PyPI

Contents:

CHAPTER 1

Installation

1.1 Install

1.1.1 apt-get

Make sure you are using the lastest version of Raspbian:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Install `pifacecommon` (for Python 3 and 2) with the following command:

```
$ sudo apt-get install python{,3}-pifacecommon
```

You will also need to set up automatic loading of the SPI kernel module which can be done with the lastest version of `raspi-config`. Run:

```
$ sudo raspi-config
```

Then navigate to `Advanced Options`, `SPI` and select `yes`.

You may need to reboot.

1.1.2 Manually

This is a more detailed description of the installation. You will have to reboot after setting up SPI and GPIO permissions.

Building and installing

Download and install with:

```
$ git clone https://github.com/piface/pifacecommon.git
$ cd pifacecommon/
$ sudo python3 setup.py install
```

Note: Subtitute `python3` for `python` if you want to install for Python 2.

Enable the SPI module

PiFace boards communicate with the Raspberry Pi through the SPI interface. The SPI interface driver is included in the latest Raspbian distributions but is not enabled by default. You can load the SPI driver manually by running:

```
$ sudo modprobe spi-bcm2708
```

You can permanently enable it one of two ways, depending on which kernel version you're on.

- Kernel Version < 3.18 (The old way): Comment out `blacklist spi-bcm2708` line in `/etc/modprobe.d/raspi-blacklist.conf`.
- Kernel Version >= 3.18 (Device Tree): add `dtparam=spi=on` to `/boot/config/txt`

The /dev/spidev* devices should now appear but they require special privileges for the user *pi* to access them. You can set these up by adding the following udev rule to `/etc/udev/rules.d/50-spi.rules`:

```
KERNEL=="spidev*", GROUP="spi", MODE="0660"
```

Then create the spi group and add the user pi:

```
$ groupadd spi
$ gpasswd -a pi spi
```

Note: To enable other users to access SPI devices (PiFace, for example) you can add them to the `spi` group with `gpasswd -a otheruser spi`.

Enable GPIO access

Interrupts work by monitoring the GPIO pins. You'll need to give the user *pi* access to these by adding the following udev rule (all on one line) to `/etc/udev/rules.d/51-gpio.rules`:

```
SUBSYSTEM=="gpio*", PROGRAM="/bin/sh -c 'chown -R root:gpio /sys/class/gpio && chmod -R 770 /sys/clas
```

Then create the gpio group and add the user pi:

```
$ groupadd gpio
$ gpasswd -a pi gpio
```

1.2 Uninstall

```
$ sudo apt-get remove python{,3}-pifacecommon
```

CHAPTER 2

Example

Here are some examples of how to use pifacecommon.

2.1 MCP23S17

```
>>> import pifacecommon.mcp23s17
>>> mcp = pifacecommon.mcp23s17.MCP23S17()
>>> mcp.gpioa.value = 0xAA
>>> mcp.gpioa.value
170
>>> mcp.gpioa.bits[3].value
1
```

CHAPTER 3

Reference

3.1 Core

pifacecommon.core.**get_bit_mask**(*bit_num*)

Returns as bit mask with bit_num set.

Parameters **bit_num** (*int*) – The bit number.

Returns int – the bit mask

Raises RangeError

```
>>> bin(pifacecommon.core.get_bit_mask(0))
1
>>> pifacecommon.core.get_bit_mask(1)
2
>>> bin(pifacecommon.core.get_bit_mask(3))
'0b1000'
```

pifacecommon.core.**get_bit_num**(*bit_pattern*)

Returns the lowest bit num from a given bit pattern. Returns None if no bits set.

Parameters **bit_pattern** (*int*) – The bit pattern.

Returns int – the bit number

Returns None – no bits set

```
>>> pifacecommon.core.get_bit_num(0)
None
>>> pifacecommon.core.get_bit_num(0b1)
0
>>> pifacecommon.core.get_bit_num(0b11000)
3
```

pifacecommon.core.**sleep_microseconds**(*microseconds*)

Sleeps for the given number of microseconds.

Parameters **microseconds** (*int*) – Number of microseconds to sleep for.

3.2 SPI

class pifacecommon.spi.**SPIDevice**(*bus=0*, *chip_select=0*, *spi_callback=None*)

An SPI Device at /dev/spi<bus>.<chip_select>.

spisend(*bytes_to_send*)
Sends bytes via the SPI bus.

Parameters **bytes_to_send** (*bytes*) – The bytes to send on the SPI device.

Returns bytes – returned bytes from SPI device

Raises InitError

3.3 Interrupts

class pifacecommon.interrupts.**EventQueue**(*pin_function_maps*)
Stores events in a queue.

add_event(*event*)
Adds events to the queue. Will ignore events that occur before the settle time for that pin/direction. Such events are assumed to be bouncing.

class pifacecommon.interrupts.**FunctionMap**(*callback*, *settle_time=None*)
Maps something to a callback function. (This is an abstract class, you must implement a SomethingFunctionMap).

class pifacecommon.interrupts.**GPIOInterruptDevice**
A device that interrupts using the GPIO pins.

gpio_interrupts_disable()
Disables gpio interrupts.

gpio_interrupts_enable()
Enables GPIO interrupts.

class pifacecommon.interrupts.**InterruptEvent**(*interrupt_flag*, *interrupt_capture*, *chip*, *timestamp*)
An interrupt event containting the interrupt flag and capture register values, the chip object from which the interrupt occured and a timestamp.

class pifacecommon.interrupts.**PinFunctionMap**(*pin_num*, *direction*, *callback*, *settle_time*)
Maps an IO pin and a direction to callback function.

class pifacecommon.interrupts.**PortEventListener**(*port*, *chip*, *return_after_kbdint=True*)
Listens for port events and calls the registered functions.

```
>>> def print_flag(event):
...     print(event.interrupt_flag)
...
>>> port = pifacecommon.mcp23s17.GPIOA
>>> listener = pifacecommon.interrupts.PortEventListener(port)
>>> listener.register(0, pifacecommon.interrupts.IODIR_ON, print_flag)
>>> listener.activate()
```

activate()
When activated the PortEventListener will run callbacks associated with pins/directions.

deactivate()
When deactivated the PortEventListener will not run anything.

deregister(*pin_num=None*, *direction=None*)
De-registers callback functions

Parameters

- **pin_num** (*int*) – The pin number. If None then all functions are de-registered
- **direction** – The event direction. If None then all functions for the given pin are de-registered

:type direction:int

register(*pin_num*, *direction*, *callback*, *settle_time=0.02*)
Registers a pin number and direction to a callback function.

Parameters

- **pin_num** (*int*) – The pin pin number.
- **direction** (*int*) – The event direction (use: IODIR_ON/IODIR_OFF/IODIR_BOTH)
- **callback** (*function*) – The function to run when event is detected.
- **settle_time** (*int*) – Time within which subsequent events are ignored.

pifacecommon.interrupts.**bring_gpio_interrupt_into_userspace**()
Bring the interrupt pin on the GPIO into Linux userspace.

pifacecommon.interrupts.**deactivate_gpio_interrupt**()
Remove the GPIO interrupt pin from Linux userspace.

pifacecommon.interrupts.**handle_events**(*function_maps*, *event_queue*, *event_matches_function_map*, *terminate_signal*)
Waits for events on the event queue and calls the registered functions.

Parameters

- **function_maps** (*list*) – A list of classes that have inheritted from FunctionMaps describing what to do with events.
- **event_queue** (multiprocessing.Queue) – A queue to put events on.
- **event_matches_function_map** (*function*) – A function that determines if the given event and FunctionMap match.
- **terminate_signal** – The signal that, when placed on the event queue, causes this function to exit.

pifacecommon.interrupts.**set_gpio_interrupt_edge**(*edge='falling'*)
Set the interrupt edge on the userspace GPIO pin.

Parameters **edge** (*string*) – The interrupt edge ('none', 'falling', 'rising').

pifacecommon.interrupts.**wait_until_file_exists**(*filename*)
Wait until a file exists.

Parameters **filename** (*string*) – The name of the file to wait for.

pifacecommon.interrupts.**watch_port_events**(*port*, *chip*, *pin_function_maps*, *event_queue*, *return_after_kbdint=False*)
Waits for a port event. When a port event occurs it is placed onto the event queue.

Parameters

- **port** (*int*) – The port we are waiting for interrupts on (GPIOA/GPIOB).
- **chip** (pifacecommon.mcp23s17.MCP23S17) – The chip we are waiting for interrupts on.
- **pin_function_maps** (*list*) – A list of classes that have inheritted from FunctionMaps describing what to do with events.

- **event_queue** (multiprocessing.Queue) – A queue to put events on.

3.4 MCP23S17

Consult the datasheet for more information.

class pifacecommon.mcp23s17.**MCP23S17**(*hardware_addr=0*, *bus=0*, *chip_select=0*)
Microchip's MCP23S17: A 16-Bit I/O Expander with Serial Interface.

Attribute iodira/iodirb – Controls the direction of the data I/O.

Attribute ipola/ipolb –This register allows the user to configure the polarity on the corresponding GPIO port bits.

Attribute gpintena/gpintenb – The GPINTEN register controls the interrupt-onchange feature for each pin.

Attribute defvala/defvalb –The default comparison value is configured in the DEFVAL register.

Attribute intcona/intconb –The INTCON register controls how the associated pin value is compared for the interrupt-on-change feature.

Attribute iocon –The IOCON register contains several bits for configuring the device.

Attribute gppua/gppub –The GPPU register controls the pull-up resistors for the port pins.

Attribute intfa/intfb –The INTF register reflects the interrupt condition on the port pins of any pin that is enabled for interrupts via the GPINTEN register.

Attribute intcapa/intcapb – The INTCAP register captures the GPIO port value at the time the interrupt occurred.

Attribute gpioa/gpiob – The GPIO register reflects the value on the port.

Attribute olata/olatb – The OLAT register provides access to the output latches.

clear_interrupts(*port*)
Clears the interrupt flags by 'read'ing the capture register.

read(*address*)
Returns the value of the address specified.

Parameters **address** (*int*) – The address to read from.

read_bit(*bit_num*, *address*)
Returns the bit specified from the address.

Parameters

- **bit_num** (*int*) – The bit number to read from.
- **address** (*int*) – The address to read from.

Returns int – the bit value from the address

write(*data*, *address*)
Writes data to the address specified.

Parameters

- **data** (*int*) – The data to write.
- **address** (*int*) – The address to write to.

write_bit(*value*, *bit_num*, *address*)
Writes the value given to the bit in the address specified.

Parameters

- **value** (*int*) – The value to write.
- **bit_num** (*int*) – The bit number to write to.
- **address** (*int*) – The address to write to.

class pifacecommon.mcp23s17.**MCP23S17Register**(*address*, *chip*)
An 8-bit register inside an MCP23S17.

class pifacecommon.mcp23s17.**MCP23S17RegisterBase**(*address*, *chip*)
Base class for objects on an 8-bit register inside an MCP23S17.

class pifacecommon.mcp23s17.**MCP23S17RegisterBit**(*bit_num*, *address*, *chip*)
A bit inside register inside an MCP23S17.

class pifacecommon.mcp23s17.**MCP23S17RegisterBitNeg**(*bit_num*, *address*, *chip*)
A negated bit inside register inside an MCP23S17.

class pifacecommon.mcp23s17.**MCP23S17RegisterNeg**(*address*, *chip*)
An negated 8-bit register inside an MCP23S17.

class pifacecommon.mcp23s17.**MCP23S17RegisterNibble**(*nibble*, *address*, *chip*)
An 4-bit nibble inside a register inside an MCP23S17.

class pifacecommon.mcp23s17.**MCP23S17RegisterNibbleNeg**(*nibble*, *address*, *chip*)
A negated 4-bit nibble inside a register inside an MCP23S17.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

p

pifacecommon.core, 7
pifacecommon.interrupts, 8
pifacecommon.mcp23s17, 10
pifacecommon.spi, 7

A

activate() (pifacecommon.interrupts.PortEventListener method), 8
add_event() (pifacecommon.interrupts.EventQueue method), 8

B

bring_gpio_interrupt_into_userspace() (in module pifacecommon.interrupts), 9

C

clear_interrupts() (pifacecommon.mcp23s17.MCP23S17 method), 10

D

deactivate() (pifacecommon.interrupts.PortEventListener method), 8
deactivate_gpio_interrupt() (in module pifacecommon.interrupts), 9
deregister() (pifacecommon.interrupts.PortEventListener method), 8

E

EventQueue (class in pifacecommon.interrupts), 8

F

FunctionMap (class in pifacecommon.interrupts), 8

G

get_bit_mask() (in module pifacecommon.core), 7
get_bit_num() (in module pifacecommon.core), 7
gpio_interrupts_disable() (pifacecommon.interrupts.GPIOInterruptDevice method), 8
gpio_interrupts_enable() (pifacecommon.interrupts.GPIOInterruptDevice method), 8
GPIOInterruptDevice (class in pifacecommon.interrupts), 8

H

handle_events() (in module pifacecommon.interrupts), 9

I

InterruptEvent (class in pifacecommon.interrupts), 8

M

MCP23S17 (class in pifacecommon.mcp23s17), 10
MCP23S17Register (class in pifacecommon.mcp23s17), 11
MCP23S17RegisterBase (class in pifacecommon.mcp23s17), 11
MCP23S17RegisterBit (class in pifacecommon.mcp23s17), 11
MCP23S17RegisterBitNeg (class in pifacecommon.mcp23s17), 11
MCP23S17RegisterNeg (class in pifacecommon.mcp23s17), 11
MCP23S17RegisterNibble (class in pifacecommon.mcp23s17), 11
MCP23S17RegisterNibbleNeg (class in pifacecommon.mcp23s17), 11

P

pifacecommon.core (module), 7
pifacecommon.interrupts (module), 8
pifacecommon.mcp23s17 (module), 10
pifacecommon.spi (module), 7
PinFunctionMap (class in pifacecommon.interrupts), 8
PortEventListener (class in pifacecommon.interrupts), 8

R

read() (pifacecommon.mcp23s17.MCP23S17 method), 10
read_bit() (pifacecommon.mcp23s17.MCP23S17 method), 10
register() (pifacecommon.interrupts.PortEventListener method), 9

S

set_gpio_interrupt_edge() (in module pifacecommon.interrupts), 9
sleep_microseconds() (in module pifacecommon.core), 7
SPIDevice (class in pifacecommon.spi), 7
spisend() (pifacecommon.spi.SPIDevice method), 7

W

wait_until_file_exists() (in module pifacecommon.interrupts), 9
watch_port_events() (in module pifacecommon.interrupts), 9
write() (pifacecommon.mcp23s17.MCP23S17 method), 10
write_bit() (pifacecommon.mcp23s17.MCP23S17 method), 10