YARPIE

YET ANOTHER RASPBERRY PI EMULATOR

by blbltheworm

August 2017 – version 0.0.2

CONTENTS

1 PREFACE – WHY ANOTHER EMULATOR?

I got my first Pi a while ago and I am still having a lot of fun discovering its capabilities and using it in various (more or less) useful projects. Unfortunately I spend quite some time sitting in trains and tinkering with a Pi is not a very mobile hobby. Thus, I was always looking for a way to work on my projects while away from home. I am aware that there already are some projects out there to emulate the GPIO port of a Pi on a standard PC but I always missed a few features. Therefore I created YARPie which is able to...

- ...emulate all features of RPi.GPIO 0.6.3 including event handling and pwm.
- ...emulate i2c-devices and allows you to generate manual or automated replies if a script tries to read data from an i2c-device.
- ...minimize the changes to your script, if switching between the emulated and the real Pi. (Just change two lines of code or write your script in a way that it automatically detects whether it is running on a Pi or a PC).
- ...provide all features within an easy to use GUI.

Currently the GPIO port of a Raspberry Pi rev. 3 (A+/B+ and above, i.e. 40 pins) is emulated and the GUI is written in pygame. If enough people are interested in this project other revisions of the board and a GUI based on Tk will be implemented (pygame proved to have some drawbacks). Maybe it would also be possible to extend my approach to GPIO Zero. So I hope this little module will be as useful to you, as it is to me. I am always happy about feedback, suggestions, bug reports... Just feel free to contact me via

- Email: blbltheworm@protonmail.com
- Git hub: https://github.com/blbltheworm/yarpie

or just look for blbltheworm in the German or English Raspberry Pi Forum.

2 INSTALLATION INSTRUCTIONS

The YARPie GUI is based on pygame, so the pygame package has to be installed. YARPie comes in a version for Pyton 2.x and Python 3.x so choose the one you prefer.
To use the emulator you have to copy the folder "RPi_emu" to the directory of your python script or the python-path (e.g. "/usr/local/lib/python2.7/dist-packages/" or "/usr/local/lib/python3.4/dist-packages/" or the corresponding folder of your python-version).

3 HOW TO USE YARPIE

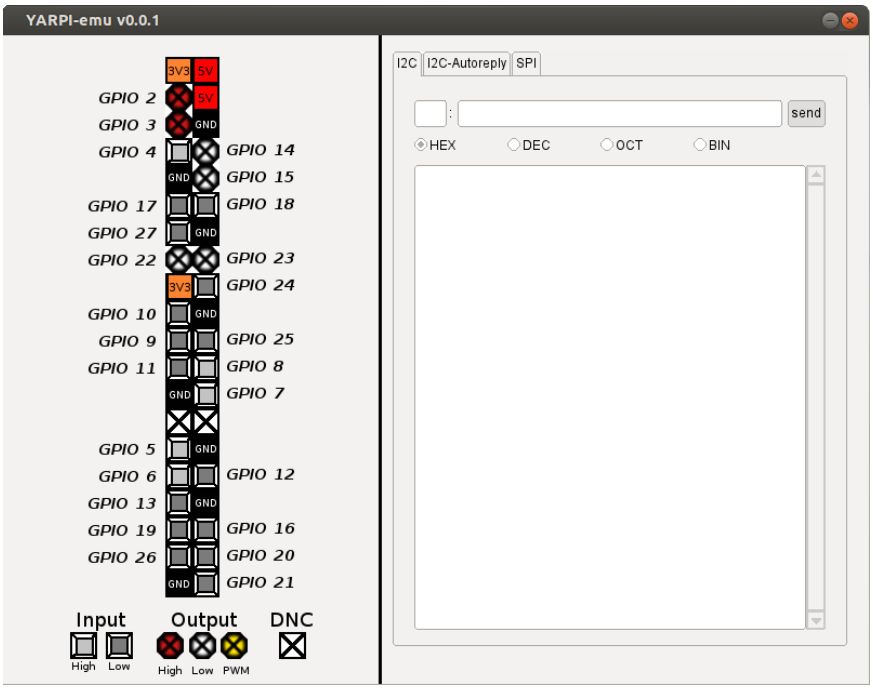
I tried my best to make this emulator as easy and convenient to use as possible. The following examples will illustrate how to work with YARPie.

3.1 EMULATING GPIO-PINS

Lets start with a classic GPIO Input/Output example. The following script defines an output at GPIO 23 as well as an input at GPIO 24. If GPIO 24 is HIGH the output will go HIGH, if GPIO 24 is LOW the output will go LOW.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(24, GPIO.IN)

while(1):
    GPIO.output(23, GPIO.input(24))
    time.sleep(0.1)
```

All you need to do to run this script with YARPie is replacing **`import RPi.GPIO as GPIO`** by **`import RPi_emu.GPIO as GPIO`**. Run the script and the emulator GUI will start automatically.



If you just like to use the GPIO port you only need the left part of the emulator window. It shows a graphical representation of the 40 pin header used in the models B+/A+ and above. At the moment only this revision of the GPIO header (revision 3) is implemented.

The GPIO pins are labelled and icons represent the possible states of the pins. The inputs (represented by small buttons) are interactive, i. e. they change their state from LOW to HIGH or vice versa, if you click on them. Outputs (represented by the classic symbol for lights) are controlled by your code.

So when clicking on the icon of GPIO 24 in this example GPIO 23 will change from LOW (white) to HIGH (red). To end the emulation close the GUI and stop your script.

As all functions of RPi.GPIO (v0.6.3) are implemented feel free to try all the examples given in the RPi.GPIO wiki to get used to YARPie.

3.2 EMULATING AN I2C-DEVICE

If your project includes an i2c-device you are also able to emulate its behaviour in YARPie. To use the i2c-emulation you only have to change two lines of code in your original script:

1. You have to replace **`import smbus`** by **`import RPi_emu.GPIO as GPIO`**.
2. You need to load the class **`SMBus`** from **`RPi_emu.GPIO`** instead of **`smbus`**, so you have to replace **`smbus.SMBus(1)`** by **`GPIO.SMBus(1)`**.

Of course if your project only uses the i2c-port and RPi.GPIO is not used you may simply replace **`import smbus`** by **`import RPi_emu.GPIO as smbus`** and everything will work fine.

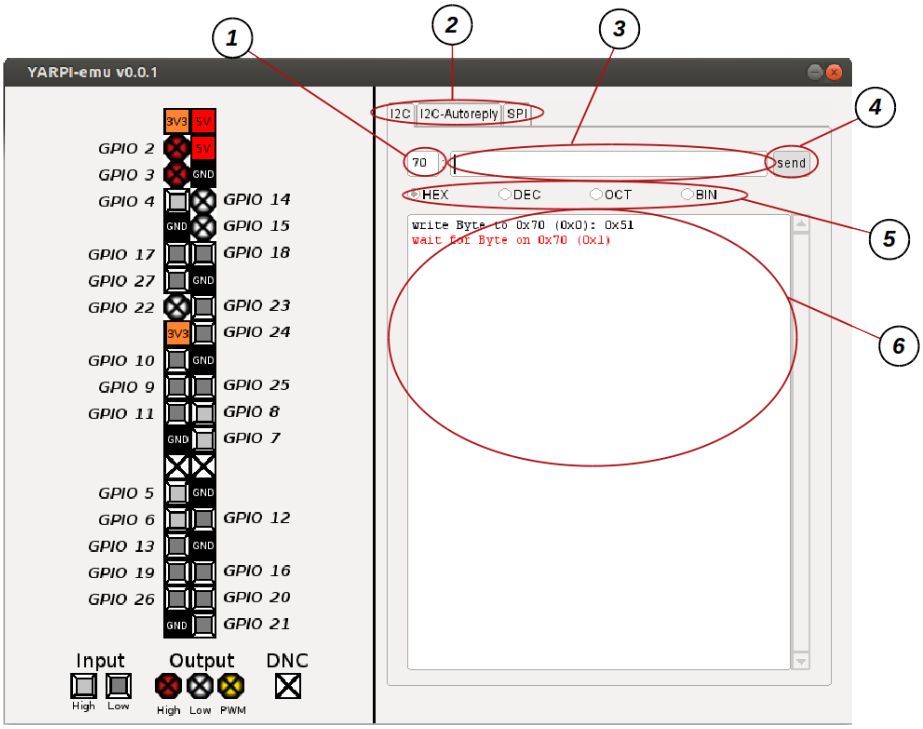
Before staring with an example one last comment about working with the emulator in projects using i2c-devices: As the emulator has no real i2c-port you can connect devices to it does not differentiate between different buses. Thus, there is no difference between **`SMBus(0)`** and **`SMBus(1)`** in YARPie but it will make a difference on a real Pi.

But lets dive into a small example. The following code was taken from http://www.instructables.com/id/Raspberry-Pi-I2C-Python/ and demonstrates the usage of a SRF08 Range Sensor.

```
import smbus
import time
bus = smbus.SMBus(0)
address = 0x70

#SRF08 REQUIRES 5V

def write(value):
    bus.write_byte_data(address, 0, value)
    return -1

def lightlevel():
    light = bus.read_byte_data(address, 1)
    return light

def range():
    range1 = bus.read_byte_data(address, 2)
    range2 = bus.read_byte_data(address, 3)
    range3 = (range1 << 8) + range2
    return range3

while True:
    write(0x51)
    time.sleep(0.7)
    lightlvl = lightlevel()
    rng = range()
    print(lightlvl)
    print(rng)
```

So first of all replace **`import smbus`** by **`import RPi_emu.GPIO as GPIO`**, replace **`bus = smbus.SMBus(0)`** by **`bus = GPIO.SMBus(0)`** and run the script. Once again the YARPie-GUI starts automatically. For i2c emulation only the

right side of the window is of importance.



The list in **(6)** shows the communication between your code and the emulated i2c devices. Values in this field are given as hex numbers. Thus, in this example the first element of the list shows that your code wrote the value 0x51 to register 0x00 of device 0x70. The second element (in red) tells you that your scripts tries to read a byte from register 0x01 of the same device (0x70), so you have to provide a value to it. There are three ways to correspond to a read command. The simplest way is to provide a manual reply via the GUI. Therefore you have to ensure that the correct device number is given in the device field **(1)**. You can type your reply into the data field **(3)** and send it to your code by clicking on send **(4)**. By standard all values typed into **(1)** and **(3)** are interpreted as hex values. You can use the radio buttons **(5)** to change between hexadecimal, decimal, octal and binary. Thus, if you type "FF" into field **(3)** and click on send a value of 255 will be send back to your code. The result of the reply to the `read_byte_data()`-function will now be shown in **(6)**. If you provide an invalid value or the wrong device number YARPie will ask you again to provide a value. Note: The radio buttons **(5)** do not influence the representation of the communication given in **(6)**, values there are always hexadecimal.

As this example also tries to read data from register 2 and 3 you have to provide values as reply from your i2c device once more. For example if you provide 5 for register 2 and 0 for register 3 the python script will print a lightlevel of 255 (the value provided for register 1) and a range of 1280

(derived from the values provided for register 2 and 3).

As this example uses an infinite loop to constantly pull data from the range sensor everything return to start and you are once more asked for a value from register 1. To overcome the problem of needing to provide values constantly it is possible to predefine replies. These are send to your script as soon as it tries to read data from the defined device + register. The next section explains how these auto replies work.

3.3 WORKING WITH AUTO REPLIES

As the example in 3.2 demonstrates it is not reasonable to provide the data to **`read_byte_data()`** & Co manually every time a read function is called. Therefore YARPie is able to provide predefined auto replies to read commands. There are two ways to do this. Either via the GUI or within your python code. Lets begin with the first option.

Start the python script once more an use the panel **(2)** to go to I2C-Autoreply. This panel slightly differs from the I2C-panel. The most important difference is an additional input field between the device and the data field. This additional field is used to provide the register from which the data will be provided. So if you like to give a value of 5 as a reply to your python code every time it tries to get the lightlevel (register 1) type in 70 as device id, 1 as register and 5 as data. Again in standard mode all values are hex values. As soon as you click on "add" the auto reply will be added to the list of auto replies in the I2C-Autoreply panel. To remove one of them just right click on the corresponding item in the list.

As your script is paused while waiting for a reply from your emulated i2c-device nothing happens after adding the auto reply. Nevertheless if you provide some data to **`read_byte_data()`** the python script will proceed and the next time your script tries to read from device 0x70, register 0x01 the emulator automatically sends a value of 0x05 back to the python script.

As mentioned above there is a second even more convenient way to create an auto reply. You do not have to create the auto replies manually every time you start your script. You can also create them within your script using **`GPIO.add_autoreply(device, register, data)`**. If you like to remove an auto reply you may either remove it via the GUI or by calling **`GPIO.remove_autoreply(device, register)`**. You can call these functions from any point within your script. If you call **`add_autoreply()`** for the same device + register combination twice the old reply will be replaced with the new data.

Before closing this section there is one last thing you need to know about auto replies: Adding an auto reply via the GUI you are only able to pro-

vide a single value per register. On the other hand the data provided via **`add_autoreply()`** may either be a single value or a list of values. If you provide a list and use **`read_i2c_block_data()`** the complete list will be returned. If you use **`read_byte_data()`** or **`read_word_data()`** only the first element of the list will be returned. If this value is larger than one byte (or one word respectively) the first byte (or the first two bytes) of this value will be returned.

The following example illustrates this behaviour. If you run it it will return 0xAB, 0xFFAB and [65451, 18, 4096].

```
import RPi_emu.GPIO as GPIO

bus = GPIO.SMBus(1)
GPIO.add_autoreply(0x50, 0x01,
    [0xFFAB, 0x0012, 0x1000])

print("0x%X" % bus.read_byte_data(0x50, 0x01))
print("0x%X" % bus.read_word_data(0x50, 0x01))
print(bus.read_i2c_block_data(0x50, 0x01))
```

3.4 DIFFERENCES BETWEEN RPI.GPIO/SMBUS AND YARPIE

While all functions of RPi.GPIO and the read and write functions of smbus are included in YARPie there are a few technical differences between your real and your emulated Raspberry Pi:

1. **`GPIO.cleanup()`**: While on a real Pi **`GPIO.cleanup()`** can be used to reset individual channels the YARPie-version will end the emulator and close the emulator-window.
2. PUD_UP/PUD_DOWN: As YARPie does not emulate a full electrical circuit the pull-up/-down resistors of the Raspberry Pi are not included in the emulation. You may still use PUD_DOWN and PUD_UP with **`GPIO.setup()`** but this will only result in defining the initial state (PUD_DOWN = LOW or PUD_UP = HIGH) of the addressed input.
3. smbus: YARPie does not differentiate between the two i2c buses of a real Pi. Thus the bus you provide calling GPIO.SMBus() is a dummy which is only used to be in better agreement with the original smbus-module. **`SMBus(0)`** and **`SMBus(1)`** will result in the same behaviour of your code. Nevertheless on a real Raspberry Pi it is important that you address the correct i2c bus.

4 ADDITIONAL TIPS

As shown before YARPie requires a minimum of adjustments to a python script designed for the Raspberry Pi to be tested on a PC. It is even possible to write a scripts in a manner that no changes are necessary. For example if you do not want to replace the import line of the RPi.GPIO module every time you test your script on your PC you could use the following lines of code:

```
try:
    import RPi.GPIO as GPIO
    print("Running on a Raspberry Pi.")
except ImportError:
    import RPi_emu.GPIO as GPIO
    print("Running in YARPie.")
```

If you like to use **`GPIO.add_autoreply()`** check GPIO.RPI_INFO and only create your auto replies if running YARPie:

```
if GPIO.RPI_INFO == 'GPIO-emu':
    GPIO.add_autoreply(0x50, 0x01, 0x01)
```

5 GET IN TOUCH

As said before: I hope this little module will be as useful to you, as it is to me. I am always happy about feedback, suggestions, bug reports... Just feel free to contact me via

- Email: ...@gmx.de
- The raspberry pi forum: ...
- Git hub: ...