Raspberry Pi

Table des matières

1 Installation

Pour installation, il faut copier rasbian sur une clé Usb.

```
$ dd bs=1M if=2018-06-27-raspbian-stretch.img of=/dev/sdg
```

Pour mettre en place SSH il faut ajouter un fichier :

```
$ mount /dev/sdg1 /mnt/raspi/boot
$ touch /mnt/raspi/boot/ssh
$ umount /dev/sdg1
```

Retrouver l'IP affectée par DHCP :

```
# tcpdump ... // ou
$ cat /var/lib/dhcp/ducpd.leases
$ ssh pi@192.168.2.14 // passwd raspberry
```

Enfin, réitérer l'opération pour installer raspbian sur la carte SD.

```
$ scp .../raspbian_latest.zip .
# unzip -p raspbian_latest.zip > /dev/mmcblk0
# mount /dev/mmcblk0p1 /mnt
# touch /mnt/ssh
# umount /mnt
# reboot
```

Configurer la distribution :

```
# raspi-config
```

2 Led

```
$ pinout

// Cablage
Gnd -- LED -- GPIO

$ raspi-gpio get 3
$ raspi-gpio set 3 op pn dh  # allumé
$ raspi-gpio set 3 op pn dl  # éteint
```

3 Bouton poussoir

Attention à ne pas utiliser le 5V, et à ne pas mettre de BP sur un GPIO positionné en sortie (ça peut flinger la carte)

- https ://www.raspberrypi.org/documentation/usage/gpio/README.md
- https ://www.framboise314.fr/le-bouton-poussoir-un-composant-banal-o-combien-etonnant/

Remarque : les BP (bouton poussoirs) issus des châssis ATX n'ont pas d'état comme les interrupteurs. Le courrant passe quand on appuie sur le BP puis est coupé lorsqu'on le relâche.

3.1 Test 1

De base les GPIO 0-8 sont positonnées à 1. Pour voir un changement d'état il faut brancher le BP sur Gnd.

```
// Schéma :
Gnd -- BP -- GPIO 4

// BP relaché
$ raspi-gpio get 4
GPIO 4: level=1 fsel=0 func=INPUT

// BP appuyé
 raspi-gpio get 4
GPIO 4: level=0 fsel=0 func=INPUT
```

3.2 Test 2

Si on veut utiliser le 3.3 pour le BP (plus intuitif), il faut utiliser une GPIO positionnée à 0 par défaut.

```
$ raspi-gpio get | grep INPUT | grep 'level=0'
GPIO 9: level=0 fsel=0 func=INPUT
GPIO 10: level=0 fsel=0 func=INPUT
...

// Schéma :
3.3v -- BP -- GPIO 10

// BP relaché
$ raspi-gpio get 10
GPIO 10: level=0 fsel=0 func=INPUT

// BP appuyé
$ raspi-gpio get 10
GPIO 10: level=1 fsel=0 func=INPUT
```

3.3 Montage

Schéma : l'ajout de la résistance permet de se prémunir de la programmation accidentelle de la broche en sortie qui provoquerait un court-circuit. Prendre un interrupteur à pied (de lampe de salon) et un unterrupteur de châssis ATX. Couper la prise et relier à la fiche via un domino. Couper les fils qui vont à la lampe et souder une résistance 1K Ohm (j'ai mis du 1.5K) entre les deux fils.

Remarque : le problème avec les interucpeurs à pied (à état) c'est que le courrant passe dès que l'on commence à appuyer (avant le click) mais qu'il ne se coupe que quand on fini de relacher (après le click).

```
// schéma
3.3v -- BP -- 1k Ohm -- GPIO 10
```

3.4 Programme

```
#!/usr/bin/python2
# -*- coding:utf-8 -*-
"""
Bouton poussoir raccordé entre GPIO10 et +3.3V
(avec résistance de protection de 1k en série)
cible               : raspberry Pi
"""
#-------------------------------------------------------------------------------
# Bibliothèques
# # pip install gpiozero
#-------------------------------------------------------------------------------
from gpiozero import Button
from gpiozero import LED
import time
#-------------------------------------------------------------------------------

bp = Button(pin=10, pull_up=False, bounce_time=.1)
led = LED(3)

if __name__ == '__main__':
    try:
        while True:
            if bp.is_pressed:
                print("BP appuyé")
                #led.on()
                bp.wait_for_release()
            else:
                print("BP relaché")
                #led.off()
                bp.wait_for_press()

            led.blink(on_time=.2, off_time=.2, n=2, background=True)

    except KeyboardInterrupt:          #sortie boucle par ctrl-c
        print("\nFin du programme\n")  #IHM
        led.off()
```

4 Serveur MAO

Configuration côté ordinateur MAO : ce script revient à lancer **qjackctl**, et **slgui** puis à connecteur leur interfaces ALSA.

```
#!/bin/bash
#set -x
set -e

# GUI may be invoked too (compatible)
# $ slgui
# $ qjackctl

# stop sooperlooper
function stop
{
    echo "bye"
    [ -z "$PID_SP" ] || kill $PID_SP
    [ -z "$PID_JACK" ] || kill $PID_JACK
    trap - EXIT # do not recall me on EXIT
    exit 0
}

# start jack
function startJack
{
    jackd -T -ndefault -dalsa -dhw:0 -r48000 -p1024 -n2 &
    PID_JACK=$!
}

# star Sooperlooper (and jack)
function startSP
{
    sooperlooper &
    PID_SP=$!
    sleep 1 # wait for jackd
}

# connect $1 to $2
function connect
{
    jack_lsp -c $1 | grep -q $2 ||
jack_connect $1 $2
}

startJack
startSP
trap stop SIGINT EXIT
connect system:capture_1 sooperlooper:common_in_1
connect sooperlooper:common_out_1 system:playback_1
read -p "Press enter to stop"
```

5 Client Raspberry Pi

On utilise le protocole OSC pour envoyer les évènements depuis le Raspberry Pi vers Sooperlooper via le réseau.

- https ://pypi.org/project/pyliblo/
- http ://essej.net/sooperlooper/doc_osc.html

5.1 Client OSC

Installation des bibliothèques :

```
# apt-get install liblo-dev
# pip install cython pyliblo
```

Client simple :

```
#!/usr/bin/env python2
# coding=utf8

import liblo # send
import time  # sleep

add = "localhost", 9951

# start record for 3 seconds
liblo.send(add, "/sl/0/hit", "record")
time.sleep(3)

# stop record (start playing loop)
liblo.send(add, "/sl/0/hit", "record")
```

5.2 Serveur OSC

Attention à bien respecter le nombre et le format des arguments (ex : 'si') sinon le serveur ignore la requête.

Serveur :

```
#!/usr/bin/env python2
# coding=utf8

import liblo # send
import time  # sleep

add_self = "localhost", 8000
running = True;

class MyServer(liblo.ServerThread):

    def __init__(self):
        liblo.ServerThread.__init__(self, add_self[1])

    @liblo.make_method('/ping', 'si')
    def ping_callback(self, path, args):
        global running
        arg1, arg2 = args
        print("pong %s %i" % (arg1, arg2))
        running = False

server = MyServer()
server.start()
while running:
    time.sleep(1)
```

Client :

```
#!/usr/bin/env python2
# coding=utf8
import liblo # send
```

```
import time  # sleep

add = "localhost", 8000

# ping
liblo.send(add, "/ping", "coucou", 22)
```

5.3 IHM ncurses

Exemple minimaliste :

```
#!/usr/bin/env python2
# coding=utf8

import curses
from curses import wrapper

def main(stdscr):
    stdscr.addstr(0, 0, "Please type a char (or q to leave)", curses.A_REVERSE)

    while True:
        c = stdscr.getch()
        stdscr.addstr(2, 2, "> %c" % c, curses.A_REVERSE)

        if c == ord('q'):
            break

wrapper(main)
```

IHM via le clavier :

```
#!/usr/bin/env python2
# coding=utf8

import sys   # exit
import liblo # send
import curses
from curses import wrapper
from time import sleep

add_self = "localhost", 8000
add_sooploop = "localhost", 9951

# server is only used to check sooperlooper is launched
class MyServer(liblo.ServerThread):
    pong = False;

    def __init__(self):
        liblo.ServerThread.__init__(self, add_self[1])

        try:
            self.sooperlooper = liblo.Address(*add_sooploop)

        except liblo.AddressError as err:
            print(err)
            sys.exit()

        self.ping()
```

```
    ### my own OSC methods (IN)

    @liblo.make_method('/pong', 'ssi')
    def pong_callback(self, path, args):
        arg1, arg2, arg3 = args
        print("pong %s %s %s %i" % (path, arg1, arg2, arg3))
        self.pong = True

    ### Sooperlooper methods (OUT)
    def ping(self):
        """ Ping and ask for status """
        address = "osc.udp://%s:%i/" % (add_self[0], add_self[1])
        print("ping %s" %address)
        liblo.send(self.sooperlooper, "/ping", address, "/pong")

def record(stdscr):
    stdscr.addstr(2, 2, "> record ", curses.A_REVERSE)
    liblo.send(add_sooploop, "/sl/0/hit", "record")

def overdub(stdscr):
    stdscr.addstr(2, 2, "> overdub", curses.A_REVERSE)
    liblo.send(add_sooploop, "/sl/0/hit", "overdub")

def pause(stdscr):
    stdscr.addstr(2, 2, "> pause  ", curses.A_REVERSE)
    liblo.send(add_sooploop, "/sl/0/hit", "pause")

def main(stdscr):
    state = 0
    state2 = 0;
    stdscr.addstr(0, 0, "space, r, o, p or q to leave", curses.A_REVERSE)

    while True:
        c = stdscr.getch()

        if c == ord('q'):
            break
        elif  c == ord('r'):
            record(stdscr)
        elif  c == ord('o'):
            overdub(stdscr)
        elif  c == ord('p'):
            pause(stdscr)
            if (state2 == 0):
                state2 = state
                state = 0
            else:
                state = state2
                state2 = 0
        elif  c == ord(' '):
            if state < 2:
                record(stdscr)
            else:
                overdub(stdscr)
            state += 1
        else:
            stdscr.addstr(2, 2, "> %c ?" % c, curses.A_REVERSE)
```

```
# OSC server (checking for sooperlooper)
server = MyServer()
server.start()
sleep(1);
if (server.pong == False):
    print("Fails to connect to Sooperlooper")
    sys.exit()

# IHM ncurses
wrapper(main)
```

6 Client complet (GPIO->OSC)

Premier jet : un seul bouton et sans IHM.

```
#!/usr/bin/python2
# -*- coding:utf-8 -*-
"""
Bouton poussoir raccordé entre GPIO10 et +3.3V
(avec résistance de protection de 1k en série)
cible              : raspberry Pi
"""
from gpiozero import Button, LED
from time import sleep
import sys   # exit
import liblo # send

add_self = "192.168.2.14", 8000
add_sooploop = "192.168.2.4", 9951
#--------------------------------------------------------------------------------

# server is only used to check sooperlooper is launched
class MyServer(liblo.ServerThread):
    pong = False;

    def __init__(self):
        liblo.ServerThread.__init__(self, add_self[1])

        try:
            self.sooperlooper = liblo.Address(*add_sooploop)

        except liblo.AddressError as err:
            print(err)
            sys.exit()

        self.ping()

    ### my own OSC methods (IN)

    @liblo.make_method('/pong', 'ssi')
    def pong_callback(self, path, args):
        arg1, arg2, arg3 = args
        print("pong %s %s %i" % (path, arg1, arg2, arg3))
        self.pong = True

    ### Sooperlooper methods (OUT)
    def ping(self):
        """ Ping and ask for status """
```

```
        address = "osc.udp://%s:%i/" % (add_self[0], add_self[1])
        print("ping %s" %address)
        liblo.send(self.sooperlooper, "/ping", address, "/pong")

def record():
    liblo.send(add_sooploop, "/sl/0/hit", "record")

def overdub():
    liblo.send(add_sooploop, "/sl/0/hit", "overdub")

def pause():
    liblo.send(add_sooploop, "/sl/0/hit", "pause")

def status_change():
    global nbPress
    global prev
    rc = (prev != -1)

    status = bp.is_pressed
    if (prev == status):
        return False
    if (status == True):
        led.on()
    else:
        led.off()
    prev = status
    return rc

if __name__ == '__main__':

    bp = Button(pin=10, pull_up=False, bounce_time=None)
    led = LED(3)
    nbPress = 0
    prev = -1

    # OSC server (checking for sooperlooper)
    server = MyServer()
    server.start()
    sleep(1);
    if (server.pong == False):
        print("Fails to connect to Sooperlooper")
        sys.exit()

    try:
        if bp.is_pressed:
            print("Svp, relachez le BP avant de commencer.")
            sys.exit(1)

        print("\nDébut du programme\n")

        while True:
            if status_change():
                nbPress += 1
                if nbPress < 3:
                    record()
                else:
                    overdub()
            sleep(0.05)
```

```
except KeyboardInterrupt:          #sortie boucle par ctrl-c
    print("\nFin du programme\n")
    led.blink(on_time=.2, off_time=.2, n=2, background=True)
    sleep(1)
```