Raspberry PI Project Template

IQAudIO Cosmic Controller and OLED

Bob Rathbone Computer Consultancy

www.bobrathbone.com

For

IQaudIO Limited, UK

iqaudio.co.uk

1st of June 2018

Contents

Figures

Tables

Introduction

This document describes a development framework for the IQaudIO Cosmic Controller and an Olimex SSD1306 128 by 64-pixel OLED.

The source files provided with this project provide a basic framework for Raspberry Pi enthusiasts and constructors to develop their own projects using the IQaudIO Cosmic Controller and Olimex OLED display.

Cosmic Controller and 128 by 64 pixel OLED display

The Cosmic controller can be used as an accessory for the Pi-Zero, the full-size Pi and even added to the top of the IQaudIO Pi-DAC+ / Pi-DAC PRO.

Features

The IQaudIO Cosmic controller consists of the following:

- A three push-button interface
- A rotary encoder
- Three status LEDs
- A 128 by 64 pixel OLED display (I2C interface) from Olimex
- Optional IR detector (Ordered separately)

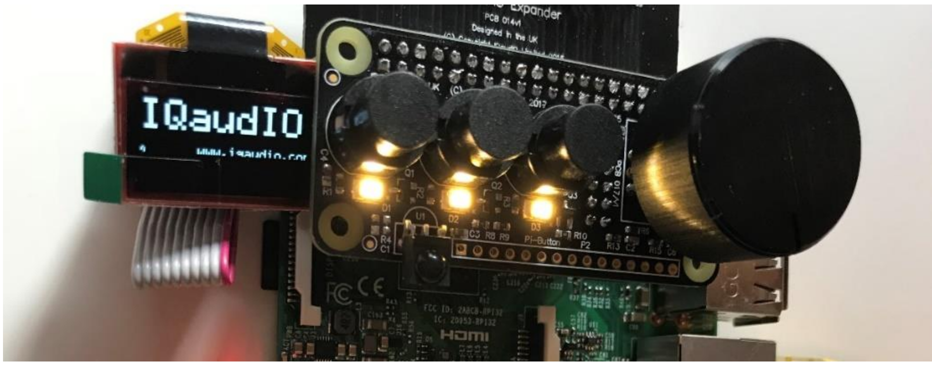


Figure 1 IQaudIO Cosmic controller and OLED display

Ordering

The Cosmic Controller can be ordered from either as a kit or pre-soldered as shown above. The IR sensor and Olimex OLED must be ordered separately. Order the parts from:

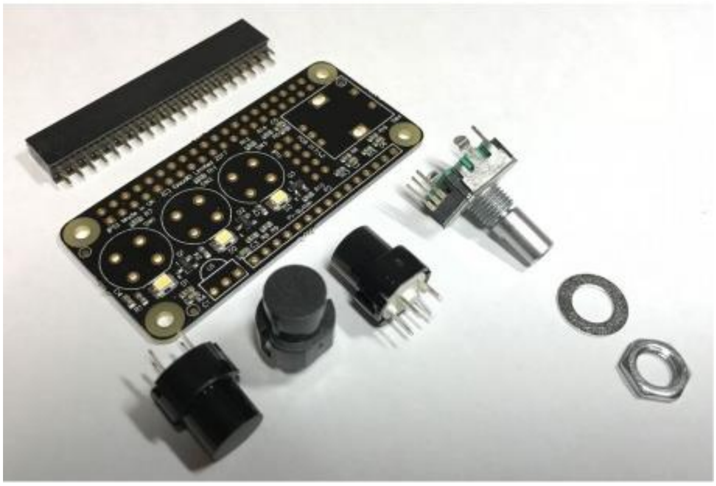


Figure 2 Cosmic controller parts kit

Construction

Cosmic controller card

The source and basic construction details are available from the following web site:
http://www.bobrathbone.com

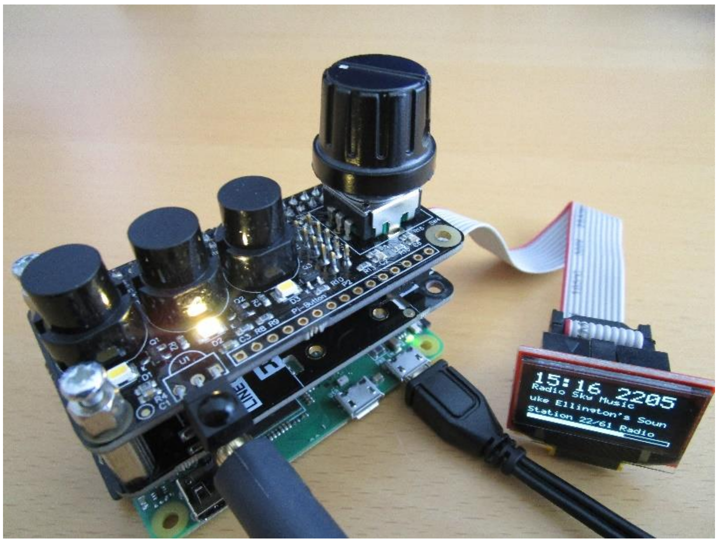


Figure 3 Raspberry Pi ZeroW and Cosmic Controller plus OLED

Connecting the Olimex display

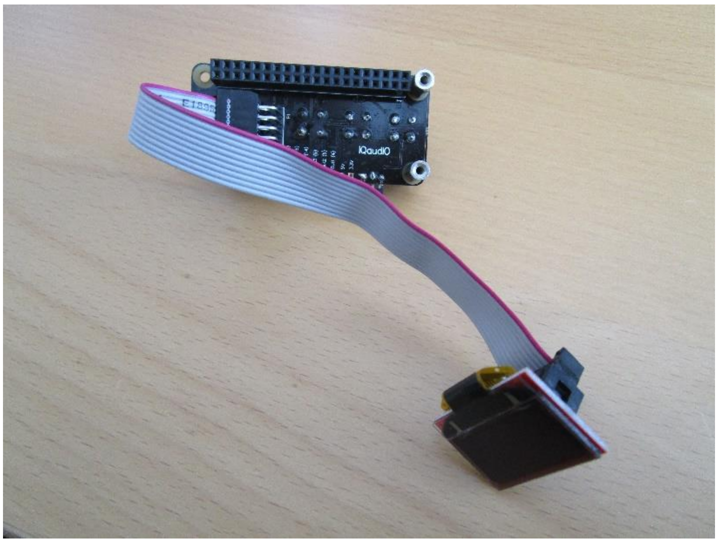


Figure 4 Connecting the Olimex OLED display

The Olimex OLED connects via a 10-pin male header on the controller board. The red wire must be connected to pin <u>one</u> on the header. The red wire should be the nearest wire to the 40-pin connector when viewing the board from below as shown in the above diagram.

Note: If constructing from a kit make sure that you connect the push switches the correct way around. The flat part of the switches should be facing the three yellow status LEDs.

Fitting the optional Infra-Red sensor is covered in the section called *Installing an IR sensor and remote control* on page 15.

Conventions used in this tutorial

Installation of the radio program requires you to enter lines at the command line prompt. This requires you to log into the Raspberry PI as user '**pi**'. The default password is **raspberry**.

Note: Don't carry out any of the following commands just yet. They are just examples.

```
Raspberrypi login: pi
Password: raspberry
pi@raspberrypi:~$ Last login: Sun Apr  6 10:18:18 2014 from 192.168.2.100
pi@raspberrypi:~$
```

The prompt line is displayed ending with a **$** sign. The **pi@raspberrypi:~** string means user 'pi' on host machine called 'raspberrypi'. The ~ character means the user 'pi' home directory **/home/pi**. In this tutorial if you are required to do something as user **pi** then only the $ sign will be shown followed by the command as shown in the example below:

```
$ sudo systemctl cosmicd status
```

You may need to be carried out as user 'root'. Usually using **sudo** is sufficient.
To become root user type in the '**sudo bash**' command:

```
$ sudo bash
root@raspberrypi:/home/pi#
```

Again, the prompt shows the username, hostname and current working directory. However only the # followed by the required command will be shown in this tutorial. For example:

```
# apt-get install libffi-dev
```

Some commands produce output which does not need to be shown. In such a case a ':' is used to indicate that some output has been omitted.

```
$ sudo systemctl status cosmicd
● cosmicd.service - Cosmic controller daemon
: {Omitted output}
May 26 11:47:01 smallpi systemd[1]: Started Cosmic controller daemon.
```

END OF EXAMPLE COMMANDS.

Installation

SD card creation

The latest version of the operating system is called **Raspbian Stretch** (Raspbian Linux 9). However almost certainly this software will work with the last release of Jessie but is untested.

Raspbian Stretch download (Tested)
http://www.raspberrypi.org/downloads

Use at least an 8 Gigabyte Card. Create an SD card running the latest version of **Raspbian Stretch/Jessie** or **Stretch/Jessie Lite**. See the *Image Installation Guides* on the above site for instructions on how to install the **Raspbian** operating system software.

Update to the latest the packages

Run the following command to update the library list.

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

Run the following command to upgrade to the latest packages for this release.

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

The above command will take some time!

If you have an older version of the Raspberry Pi then update the firmware.

```
$ sudo rpi-update
```

Reboot the Raspberry Pi.

```
$ sudo reboot
```

Important: After upgrading the system the repository locations may no longer be valid. Re-run apt-get update to refresh the package list. Failing to do this may result in packages failing to install.

Re-run the update command to update the library list.

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

The IQAudIO Cosmic controller and OLED is also require **libffi-dev** plus other necessary libraries. Carry out the following instructions.

```
$ sudo apt-get install libffi-dev
$ sudo apt-get install build-essential libi2c-dev i2c-tools python-dev
```

Enable the i2c module

Enable i2c kernel module with raspi-config. Run the following:

```
$ sudo raspi-config
```

Select option 5 Interfacing Options:

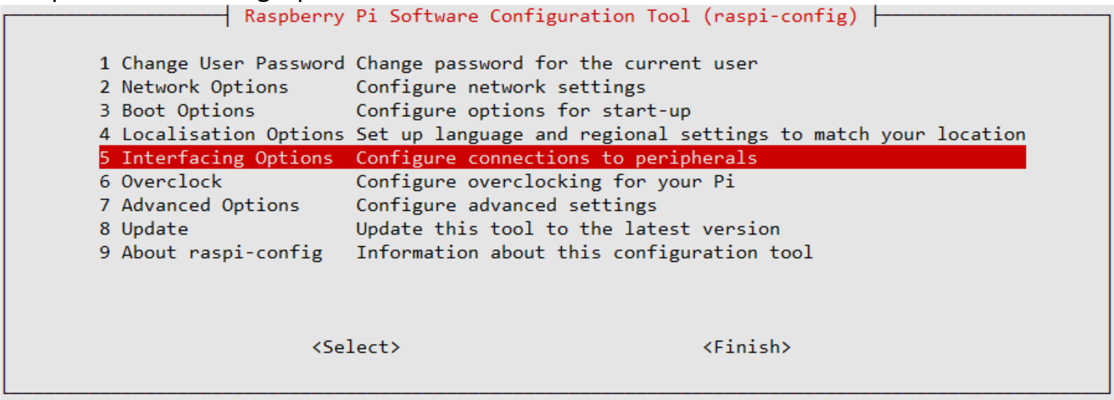


Figure 5 Running raspi-config

Select Option P5 to enable the I2C interface.

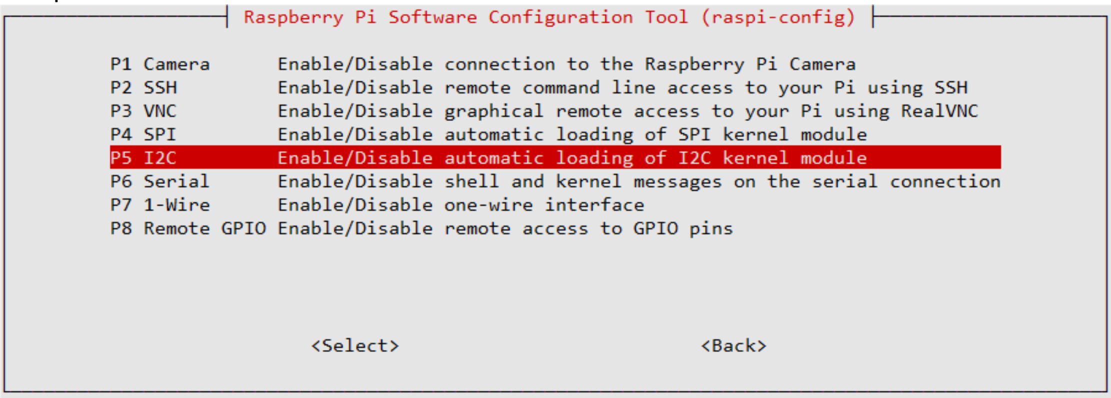


Figure 6 Select I2C interface

Enable the ARM I2C interface by selecting ‘Yes’ and press enter.

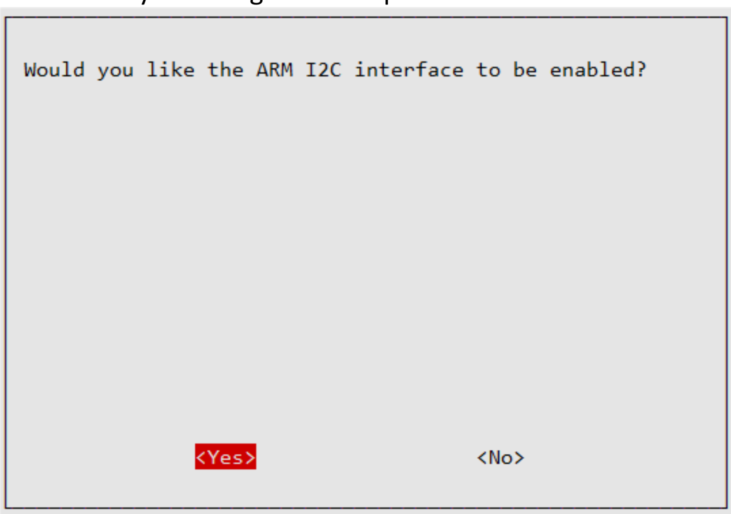
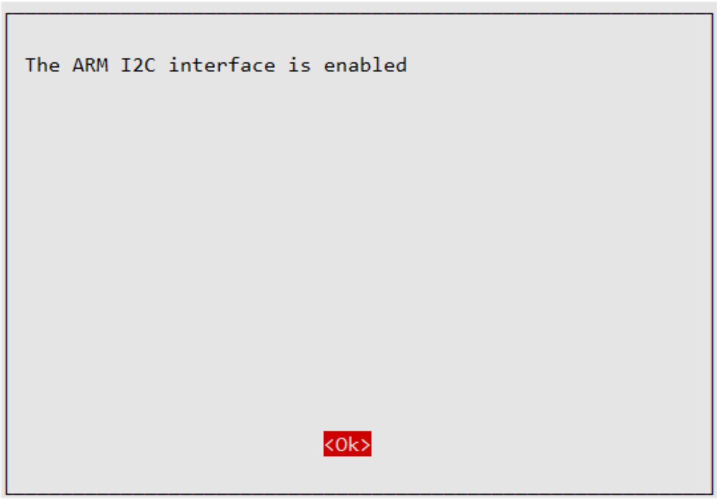


Figure 7 Enable I2C

The program now reports that the I2C interface is enabled. Press enter and then ‘finish’ to exit the program:



Important: Due to a fault in **raspi-config** in certain releases of Jessie the I2C interface is not always enabled in the **/boot/config.txt** configuration file.

Check that the following line has been enable in the **/boot/config.txt** configuration file.

```
dtparam=i2c_arm=on
```

If not edit the **/boot/config.txt** configuration file and enable it.

```
$ sudo nano /boot/config.txt
```

Reboot the system to load the I2C kernel module:

```
$ sudo reboot
```

Test the I2C interface

If you have not already done so, connect the IQAudIO Cosmic Controller and OLED to the Raspberry Pi and connect the power. Run **i2cdetect -y 1**

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

The I2C hex address of the OLED should be seen (3c). If not proceed to the Troubleshooting section on page 27.

Install the cosmicd package

Now use **wget** to the software package:

```
$ wget http://www.bobrathbone.com/raspberrypi/packages/cosmicd_1.0_armhf.deb
```

Run **dpkg** to install the package.

```
$ sudo dpkg -i cosmicd_1.0_armhf.deb
```

The **dpkg** program will install the files.

```
(Reading database ... 131542 files and directories currently
installed.Preparing to unpack cosmicd_1.0_armhf.deb ...
Unpacking cosmicd (1.0)
:
```

This will install the package in **/usr/share/cosmicd** directory.

Operation

Once installation of the hardware has been completed you are ready to run the test programs. There are five test programs supplied with the package.

- test_controller.py Test rotary encoder, buttons and LEDs
- test_oled.py Test basic OLED text functions
- test_graphics.py Test OLED graphics functions
- test_remote.py IR remote control test (See page 20)
- cosmicd.py A daemon test program for OLED and Cosmic controller

All the test programs are in the **/usr/share/cosmicd** directory

Cosmic Controller test

```
$ /usr/share/cosmicd/test_controller.py
```

The following will be displayed:

```
Test Cosmic Controller Class
============================
Left switch GPIO 4
Middle switch GPIO 5
Right switch GPIO 6
Encoder A GPIO 23
Encoder B GPIO 24
Encoder switch GPIO 27
statusLed 14 15 16
```

Operate the push buttons

```
Button pressed on GPIO 4
Button pressed on GPIO 5
Button pressed on GPIO 6
```

The LED under each button will also light up in turn.

Operate the rotary encoder:

```
Rotary event  1 CLOCKWISE
Rotary event  1 CLOCKWISE
Rotary event  2 ANTICLOCKWISE
Rotary event  2 ANTICLOCKWISE
Rotary event  3 BUTTON DOWN
```

The Rotary encoder button will switch off all of the status LEDs.

Press CTL-C to exit the program:

```
^C Stopped
```

This program does not display anything on the OLED screen. Study the code in **test_controller.py** to understand how it works.

The basic OLED test

This test displays a splash screen time and text along with a horizontal slider.

```
$ /usr/share/cosmicd/test_oled.py
```

OLED Graphics test

This test displays a splash screen, circles, squares, lines and a simple animation sequence (clowns).

```
$ /usr/share/cosmicd/test_controller.py
```

Running the Cosmic Controller daemon

The Cosmic controller daemon is designed to provide a practical example application running as a system service (also known as a daemon). It is started by **systemd** from the **/lib/systemd/system/cosmicd.service** definition.

```
$ sudo systemctl start cosmicd
```

Once running its status can be checked with the following command

```
$ sudo systemctl status cosmicd
```

If running OK it will display the following:

```
● cosmicd.service - Cosmic controller daemon
   Loaded: loaded (/lib/systemd/system/cosmicd.service; disabled; vendor
preset: enabled)
   Active: active (running) since Wed 2018-05-16 14:52:18 CEST; 45s ago
 Main PID: 574 (python)
   CGroup: /system.slice/cosmicd.service
           └─574 python /usr/share/cosmicd/cosmicd.py nodaemon
```

It can be stopped with the following command:

```
$ sudo systemctl stop cosmicd
```

Implementing this program as a system service allows the **cosmicd** daemon to be automatically started at boot time. To enable this run the following command.

```
 $ sudo systemctl enable cosmicd
Created symlink /etc/systemd/system/multi-user.target.wants/cosmicd.service
→ /lib/systemd/system/cosmicd.service.
```

Reboot the Raspberry Pi. The **cosmicd** service should start automatically and the OLED should display the Pi logo followed by the time on the top line.

The **cosmicd** service can be disabled at boot time with the following command:

```
$ sudo systemctl disable cosmicd
```

Installing an IR sensor and remote control

Note: The IR sensor is not supplied as standard but can be ordered from IQAudio.

IR Sensor

If you wish to use an IR remote control in your application then purchase an IR sensor TSOP38238 or similar. The output in this case will be connected to GPIO 11 (Pin 23) or GPIO 26 (pin 37) for LCD or Adafruit Plate respectively.

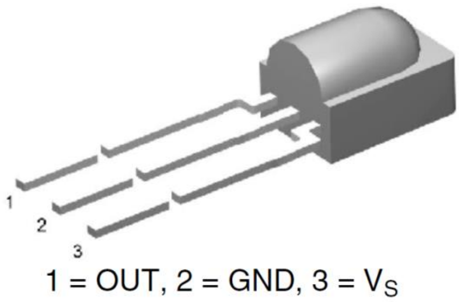


Figure 8 TSOP38238 IR sensor

The TSOP38xxx series works from 2.5 to 5.5 volts and is ideal for use the Raspberry PI.

IR sensor	Description	RPi
Pin 1	Signal Out	GPIO
Pin 2	Ground	Pin 6
Pin 3 **	Vs 3.3 Volts	Pin 1

There are equivalent devices on the market such as the TSOP4838 which operate on 3.3 volts only.

See http://www.vishay.com/docs/82491/tsop382.pdf for more information on these IR sensors.

Tip: These IR sensors are very prone to damage by heat when soldering them. It is a good idea to use a 3-pin female connector and push the legs of the IR detector into them. If you solder the IR detector directly into a circuit then take precautions by connecting a crocodile clip across each pin in turn whilst soldering it. See figure below:

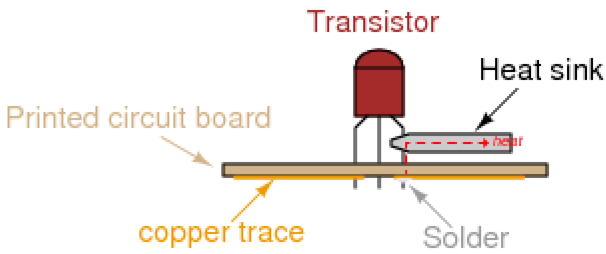


Figure 9 Soldering precautions

Remote control

Almost any surplus IR remote control can be used with this project. Later on it is explained how to set up the remote control with the radio software. You will need to install the software for IR sensor.
See the section called *Installing the Infra Red sensor software* on page 16.

Remote Control Activity LED

Pick one of the three status LEDs as the IR activity LED. The following table shows the GPIO numbers used for the LED connections.

Table 1 Remote Control Activity LED

LED	Position	GPIO	40 pin GPIO header
LED 1	Left	14	8
LED 2	Middle	15	10
LED 3	Right	16	36

How to configure the LED is shown on in the section called **Configuring the remote control** activity LED on page 21.

Installing the Infra Red sensor software

If you haven't already done so update the operating system first.

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

Install **lirc** with the following command:

```
$ sudo apt-get -y install lirc
```

```
Now you must create an /etc/lirc/lircd.conf for your remote control.
Either download one from here:

    http://lirc.sourceforge.net/remotes/

Or generate one yourself with the following command:

    sudo irrecord -f -d /dev/lirc0 /etc/lirc/lircd.conf
```

It is necessary to install the python-lirc package.

Note: Currently earlier versions **Raspbian Stretch** did not have the **python-lirc** package in its repository which meant it had to be downloaded from the author Tom Preston at GitHub. For **Rasbian Jessie** it can be installed using apt-get as shown below.

If using install **Rasbian Jessie** or latest **Stretch** install **python-lirc** with the following command:

```
$ sudo apt-get -y install python-lirc
```

If the above fails the download **python-lirc** from https://github.com/tompreston/python-lirc/releases

At the time of writing the latest release of the package was 1.2.1. To download this package run the following command:

```
$ wget https://github.com/tompreston/python-
lirc/releases/download/v1.2.1/python-lirc_1.2.1-1_armhf.deb
```

Note: The above command is all one line. Now install the software:

```
$ sudo dpkg -i python-lirc_1.2.1-1_armhf.deb
```

Now edit **/etc/lirc/lirc_options.conf** and change the driver name in the [lircd] section from 'devinput' to 'default'.

```
[lircd]
nodaemon            = False
driver              = default
```

The following step requires configuration of the **lirc-rpi dtoverlay** in **/boot/config.txt** file. Add the following line to the end of the **/boot/config.txt** file.

```
dtoverlay=lirc-rpi,gpio_in_pin=25,gpio_in_pull=high
```

Save the **/boot/config.txt**.

Now copy the **lircrc.dist** file to **/etc/lirc/lircrc**

```
$ cd /usr/share/cosmicd
$ sudo cp lircrc.dist /etc/lirc/lircrc
```

Reboot the Raspberry Pi

```
$ sudo reboot
```

Now test your remote control with the mode2 command. You should see lots of codes when you press buttons on the remote control.

```
$ sudo mode2 -d /dev/lirc0
space 5358604
pulse 4591
space 4459
pulse 651
```

Note: If the above does not work it is pointless continuing with the installation. The problem with the IR sensor and remote control must first be rectified. See Troubleshooting IR sensor problems on page 27.

If you see the following output then edit **/etc/lirc/lirc_options.conf** and change the driver name in the [lircd] section from 'devinput' to 'default' as previously shown.

```
Using driver devinput on device /dev/lirc0
Trying device: /dev/lirc0
Using device: /dev/lirc0
Running as regular user pi
Partial read 4 bytes on /dev/lirc0
```

Check that the **lirc0** device exists. If it doesn't exist you cannot continue. Check that the **/boot/config.txt** file has been correctly configured as shown above.

```
$ ls -la /dev/lirc0
crw-rw---T 1 root video 246, 0 Jan  1  1970 /dev/lirc0
```

It is now necessary generate a configuration for the remote control then first move the distributed **/etc/lirc/lircd.conf** file out of the way.

```
$ sudo mv /etc/lirc/lircd.conf /etc/lirc/lircd.conf.org
```

Now run the configuration program.

```
$ sudo irrecord -f -d /dev/lirc0 /etc/lirc/lircd.conf
```

If you see the following

```
irrecord: could not open /dev/lirc0
irrecord: default_init(): Device or resource busy
irrecord: could not init hardware (lircd running ? --> close it, check
permissions)
```

Run the following command:

```
$ sudo service lircd stop
```

If the problem persists, make sure the **/boot/config.txt** file has been correctly set up as previously shown and that a reboot was carried out.

Follow the instructions in the **irrecord** program exactly!

The **irrecord** program will ask you for the name of the file. You must give it the name **cosmicd**.

```
Checking for ambient light  creating too much disturbances.
Please don't press any buttons, just wait a few seconds...

No significant noise (received 0 bytes)

Enter name of remote (only ascii, no spaces) :cosmicd
Using cosmicd.lircd.conf as output filename
```

It will then ask for the names of the buttons that you want to configure. You may not make your own names up. You must use the names shown in the first column of the following table and which are defined in **/etc/lirc/lircrc.**

It is a good idea to just start with the basic keys for volume up and channel change and when you have the remote control working re-configure with all of the keys shown in *Table 2 Remote Control Key names and functions*.

Table 2 Remote Control Key names and functions

Key Names	Normal	Search	Source	Options
KEY_VOLUMEUP	Volume up	Volume up	Volume up	Volume up
KEY_VOLUMEDOWN	Volume down	Volume down	Volume down	Volume down
KEY_CHANNELUP	Channel up	Channel up	Channel up	Channel up
KEY_CHANNELDOWN	Channel down	Channel down	Channel down	Channel down
KEY_MUTE	Mute sound	Mute sound	Mute sound	Mute sound
KEY_MENU	Step menu	Play selected	Load tracks/stations	Next menu
KEY_UP	Not used	Previous artist	Toggle source	Previous option
KEY_DOWN	Not used	Next artist	Toggle source	Next option
KEY_LEFT	Not used	Track up	Not used	Toggle option
KEY_RIGHT	Not used	Track down	Not used	Toggle option
KEY_OK	Step menu	Play selected	Load tracks/stations	Next menu

When finished assigning names to the keys press enter. The program reports the following.

```
Successfully written config file cosmicd.lircd.conf
```

The file has been saved in the user home directory.

Now copy **cosmicd.lircd.conf** to **/etc/lirc/lircd.conf**

```
$ sudo cp /home/pi/cosmicd.lircd.conf /etc/lirc/lircd.conf
```

The actual list of available names that may be used can be displayed with the following command:

```
$ sudo irrecord --list-namespace
```

There are more than 440 key names but only use the ones defined in the list above.

The activity LED will flash a few times however it is necessary to reboot the system to enable the new IR remote configuration.

```
$ sudo reboot
```

Now test the remote control with your newly configured remote control.

Testing the remote control

If there are problems with the remote control you can test using **ircat** which will display the key codes that have been programmed. Stop the **irradiod** daemon first and restart service **lircd**.

The service name to start lirc has changed between releases. In older releases it is called **lirc** and the newer release it is called **lircd**. Use the correct name in the following commands:

```
$ sudo service lircd start
Loading LIRC modules:.
Starting remote control daemon(s) : LIRC :.
Starting execution daemon: irexec:.
```

Now run **ircat** and press each key on the remote control in turn:

```
$ ircat piradio
KEY_VOLUMEUP
KEY_VOLUMEDOWN
KEY_CHANNELUP
KEY_CHANNELDOWN
KEY_MENU
KEY_MUTE
KEY_UP
KEY_RIGHT
KEY_LEFT
KEY_DOWN
KEY_LANGUAGE
KEY_INFO
```

Use Ctrl-C to exit. If keys are not responding repeat the previous Remote-Control installation procedure. If **ircat** works OK, run the test_remote.py program.

Running the test_remote.py program

```
$ cd /usr/share/cosmicd
$ sudo ./test_remote.py
Test remote control pid 8428
Activity LED on GPIO 15
Press Ctl-C to exit
Listener on socket <module 'socket' from '/usr/lib/python2.7/socket.pyc'>
established
```

Now press the buttons on the remote control in turn.

```
Callback got button: KEY_CHANNELUP
listener: KEY_CHANNELDOWN
Callback got button: KEY_CHANNELDOWN
listener: KEY_VOLUMEUP
Callback got button: KEY_VOLUMEUP
listener: KEY_VOLUMEDOWN
Callback got button: KEY_VOLUMEDOWN
listener: KEY_VOLUMEDOWN
listener: KEY_OK
Callback got button: KEY_OK
```

The right-hand LED (GPIO 16) should also flash.

Disabling the repeat on the volume up/down buttons

If you wish to disable the repeat on the volume control the edit the **/etc/lirc/lircrc** file, set **repeat = 0**, for KEY_VOLUMEUP and KEY_VOLUMEDOWN definitions.

```
begin
  prog = piradio
  button = KEY_VOLUMEUP
  config = KEY_VOLUMEUP
  repeat = 0
end

begin
  prog = piradio
  button = KEY_VOLUMEDOWN
  config = KEY_VOLUMEDOWN
  repeat = 0
end
```

Configuring the remote control activity LED

All configuration is done in the **config_class.py** program. By default, the IR activity LED is set to LED3 (Right-hand LED). This can be changed in **config_class.py**.

```
        # Status LEDs
        _led1 = 14
        _led2 = 15
        _led3 = 16

        _ir_activity_led = _led3
```

If you do not want to show any LEDs set these definitions to 0. However, the Raspberry Pi switches on GPIO 15 on reboot. To disable this, add the following lines to **/etc/rc.local** just before the exit statement.

```
gpio -g mode 15 out
gpio -g write 15 0

exit 0
```

Building your own software

Simple programs

The whole purpose of this manual is to provide a basic framework for creating your own applications using the IQaudIO Cosmic Controller and OLED.

Any one or more of the programs below can be copied and modified to create your own software.

- test_controller.py   Test rotary encoder, buttons and LEDs
- test_oled.py   Test basic text functions on the OLED
- test_graphics.py   Test graphics functions
- test_remote.py   Test the remote-control IR sensor

Don't modify these programs directly but copy them with a new name. For example, if you want to write your own clock program called **myclock.py** then create it from **test_oled.py**:

```
$ cp -p test_oled.py myclock.py
```

Modify **myclock.py** with your own code. However, when you have finished you still have to log into the Raspberry Pi to run your program. If you want to run the program boot time you can add it to the **/etc/rc.local** file. Commands in the **/etc/rc.local** file are executed after all of the system services are loaded.

Edit the **/etc/rc.local** file and add the full pathname of your new program to it just before the exit statement.

```
/usr/share/cosmicd/myclock.py
exit 0
```

This will start **myclock.py** at boot time. However, this is not the professional way of starting a program (service) on start-up. If you want to stop the program, for example, you have to identify its process ID and issue the **kill** command to stop it. It is not very convenient. A far better way is to make your program run as a daemon which is covered in the next section.

Writing your own daemon

As time goes on you may want to create a more sophisticated program which can run as a system service otherwise known as a daemon.

The following program runs as daemon and is started by the **cosmicd.service** file in the **/lib/systemd/system** directory. By convention system service names usually end with a letter 'd' to specify that it is a daemon. However, this is not a requirement and you can use any name you wish.

- cosmicd.py   A test daemon for the Cosmic controller and OLED

System services (daemons) can be started, stopped, enable with the **systemctl** command.
For example:

sudo systemctl start cosmicd
sudo systemctl stop cosmicd
sudo systemctl enable cosmicd

A quick note on starting and stopping system services. All system services for Linux and Unix used to be started by the **System V init** program. This would call the appropriate script in the **/etc/init.d** directory. If you look in this directory you will still see legacy start-up scripts in there. The system V command for starting and stopping daemons is the **service** command. For example

sudo service cosmicd start

The above will still work however the command is intercepted by **systemd** and the startup is executed by **systemctl**.

The modern method is to use **systemd** commands as previously shown, however, both **System V** and **systemd** commands will work. From now on we will use the **systemctl** command.

Creating a new daemon

Copy the cosmicd.py program to your new program. For example, a new daemon called myprogd.py

```
$ cd /usr/share/cosmicd
$ cp cosmicd.py myprogd.py
```

Modify the myprogd.py program with your code. During testing run it with the **nodaemon** option.

```
$ sudo ./myprogd.py nodaemon
```

The **nodaemon** option stops it running as a daemon and you can put print statements into the program to de-bug it. If you run it as daemon it disconnects from the terminal. The statement below will run it as a daemon.

```
 $ sudo ./myprogd.py start
```

The following command will stop the program.

```
$ sudo ./myprogd.py stop
```

Once the program has been written you can create a service unit for it. Again, there is a template for this either already installed in **/lib/systemd/system/** or in the source directory (If you downloaded the source from GitHub).

```
$ sudo cp /lib/systemd/system/cosmicd.service  myprogd.service
```

Amend myprogd.service to point to myprogd:

```
[Service]
Type=simple
ExecStart=/usr/share/cosmicd/myprogd.py nodaemon
```

Note: Although you start **myprogd.py** with the **nodaemon** option, **systemctl** will daemonise the program.

Edit the description

```
[Unit]
Description=My program daemon
```

Now copy the new service file to the /lib/systemd/system/ directory.

```
$ sudo cp myprogd.service /lib/systemd/system/
```

Test it:

```
$ sudo systemctl start myprogd
```

Check the status:

```
$ sudo systemctl status myprogd
● cosmicd.service - My program daemon
   Loaded: loaded (/lib/systemd/system/myprogd.service; disabled; vendor
preset: enabled)
   Active: active (running) since Sat 2018-05-26 11:47:01 CEST; 12s ago
 Main PID: 1536 (python)
   CGroup: /system.slice/myprogd.service
           └─1536 python /usr/share/cosmicd/myprogd.py nodaemon

May 26 11:47:01 smallpi systemd[1]: Started My program daemon.
```

Now enable it to start it at boot time:

```
$ sudo systemctl enable myprogd
```

You should find that your new service starts the next time you start the Raspberry Pi.

Building a Debian package

You can either supply the program files via a repository such as GitHub (https://github.com) or you can create a Debian package which can be downloaded and installed using **dpkg**.

First install the package build programs.

```
$ sudo apt-get -y install equivs apt-file lintian
```

Copy build.sh to mybuild.sh and build definition files

```
$ cp build.sh mybuild.sh
$ cp cosmicd to myprogd
$ cp cosmicd.preinst myprogd.preinst
$ cp cosmicd.postinst myprogd.postinst
$ cp cosmicd.postrm myprogd.postrm
```

The **myprogd** file is the package build definition file. Edit the PKGDEF and PKG definitions in this file.

```
PKGDEF=myprogd
PKG=myprogd
```

Change all references to cosmicd in **myprogd** file to myprogd.

Change the Description: line.

Change the Version: if required

Change Maintainer: and Email-Address

Edit the Files: list to only include the files needed for your new package.

Now run the build (Do not use sudo).

```
$ ./mybuild.sh
```

Correct any errors and rebuild as required. More information about build files can be found at https://www.debian.org/doc/debian-policy/

If checking the package integrity with **lintian** you can ignore the warnings. Errors however should be corrected although probably your package will install and run OK.

Package installation

Install the package:

```
$ sudo dpkg -i myprogd_1.0_armhf.deb
```

The package should now be installed in **/usr/share/myprogd**.

Now enable the package at boot time.

```
$ sudo systemctl enable myprogd
```

Troubleshooting

Rotary encoder problems

Run the test program shown in the section called *Cosmic Controller test* on page 12. If this still doesn't work check the soldering of rotary encoder.

Button problems

If you constructed the Cosmic Controller board yourself check that the buttons are correctly orientated. The flat side of the button should be facing the three yellow LEDs.

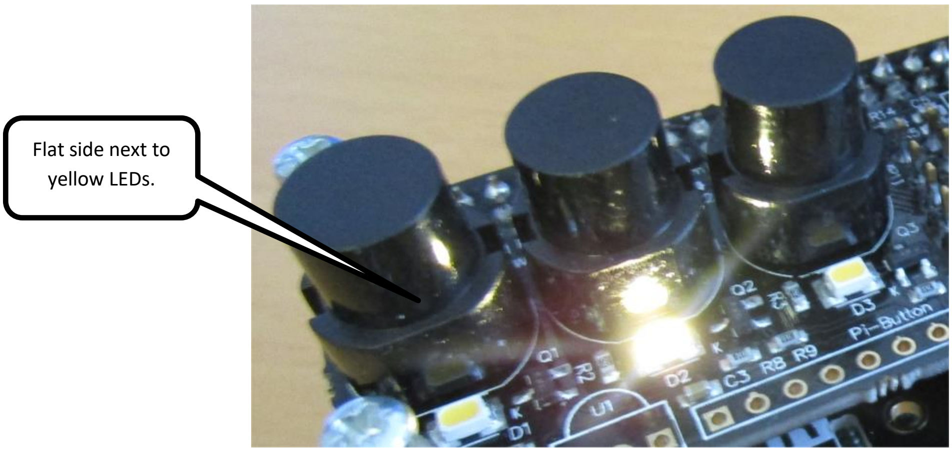


Figure 10 Button orientation

If by any chance any of the buttons have been incorrectly positioned they are almost impossible to get out in one piece without specialised equipment. Remove the incorrectly positioned switch and solder in a new one.

Olimex OLED display problems

First of all, check that the ten-pin connector has the red wire connected to pin 1 as shown in the section *Connecting the Olimex display* on page 6.

Next check that the I2C libraries are installed as shown in *Enable the i2c module* on page 9. Check that the OLED is visible at hex address 0x3C as shown in the section *Test the I2C interface* on page 10.

Re-run the OLED tests on page 13.

IR sensor problems

Causes of problems with the IR sensor can be any one of the following:

- The remote control is not working (Check batteries)
- The IR sensor has been damaged during soldering
- The IR sensor is connected up the wrong way around
- LIRC is not correctly installed.

Check the orientation of the IR sensor. See Figure 3 on page 6.
Repeat the installation procedure shown in the section Installing an IR sensor and remote control on page 15.

Programming reference

imports

To build your own software you need to begin with the following imports:

```
from rotary_class import RotaryEncoder
from status_led_class import StatusLed
from config_class import Configuration
from cosmic_class import Button
```

Defining the push buttons

Start by getting the GPIO numbers from the configuration class.

```
config = Configuration()

left_switch = config.getLeftSwitch()
middle_switch = config.getMiddleSwitch()
right_switch = config.getRightSwitch()
```

Now define the buttons. The first parameter of Button is the GPIO for the switch. The second parameter is a so-called call-back routine which will be invoked when there is a button event.

```
Button(left_switch, button_event)
Button(middle_switch, button_event)
Button(right_switch, button_event)
```

The call-back routine must be defined before the above code.

```
def button_event(gpio):
    print "Button pressed on GPIO", gpio
    “””
    Add your code here
    “””
```

You can of course define separate event handlers for each button.

```
Button(left_switch, left_button_event)
Button(middle_switch, middle_button_event)
Button(right_switch, right_button_event)
```

Defining the rotary encoder

As for the buttons get the GPIO numbers for the rotary encoder from the configuration class.

```
encoder_switch = config.getEncoderSwitch()
encoder_a = config.getEncoderA()
encoder_b = config.getEncoderB()
```

The first two parameters in this call are the GPIO numbers of the rotary encoder A and B inputs. The third parameter is the GPIO number for the rotary encoder push switch. The fourth parameter is the call-back routine which will be invoked whenever there is a rotary encoder event:

```
rotaryknob = RotaryEncoder(encoder_a,encoder_b,encoder_switch,rotary_event)
```

The rotary event routine must be defined before the above code.

```
def rotary_event(event):
        print "Rotary event ", event
        if event == RotaryEncoder.CLOCKWISE:
               # Put your code for clockwise events here

        elif event == RotaryEncoder.ANTICLOCKWISE:
               # Put your code for anti-clockwise events here

        elif event == RotaryEncoder.BUTTONDOWN:
               # Put your code for encoder button down events here
               # There is also a BUTTONUP event
```

Handling the LEDs

Define and initialise the status LEDs.

```
led1 = config.getLed1()
led2 = config.getLed2()
led3 = config.getLed3()
statusLed = StatusLed(led1,led2,led3)
```

Switching on a LED on and off:

```
statusLed.set(StatusLed.LED1,True)
statusLed.set(StatusLed.LED1,False)
```

The left-hand, middle and right-hand LEDs are defined as LED1, LED2 and LED3.

Switching off all LEDs:

```
statusLed.clear()
```

Programming the OLED

The basic routines for driving the OLED have been written by Olimex Limited. A wrapper called **oled_class.py** has been written to provide extra functionality to the basic Olimex routines. For example, scrolling and draw rectangle routines. **See test_oled.py** for an example of these routines.

The Olimex OLED is 128 pixels wide by 64 pixels high. It has eight display lines.

The X co-ordinate is from 0 to 127 and the Y co-ordinate is from 0 to 63.

All of the routines shown below carry out the required drawing but nothing appears on the screen until you call the update function.

```
oled.update()
```

To use the OLED wrapper routines first import the Oled class and define the **oled** object.

```
from oled_class import Oled
oled = Oled()
```

Clear the screen first

```
oled.clear()
```

Text routines

Set the font size 1 to 9 (only 1 to 5 are practical)

```
oled.setFont(1)
```

Display text on line 1. Line numbers are from 1 through 8. A line with font size 1 is 20 characters.

```
oled.out(1,"abcdefghijklmonopqrstuvwxyz",no_interrupt)
```

The scrolling routines require a call-back to interrupt the scrolling routine, if say a button event occurs. In the above routine we do not need to interrupt the instruction so **no_interrupt** is defined as follows:

```
def no_interrupt():
        return False
```

If you wish to display lines longer than can be displayed on a single line they will automatically be scrolled. In such a case you should define an interrupt routine as shown in the **cosmicd.py** example program.

Advice: Do not re-display a line if the text has not changed. The Olimex OLED is not a particularly fast device so performance will improve if you only display lies have changed. Again, see the **cosmicd.py** program for an example of how this can be done.

Graphics routines

Drawing the splash screen for two seconds:

```
oled.drawSplash("bitmaps/raspberry-pi-logo.bmp",2)
```

Bitmaps must be placed in the bitmaps sub-directory.

Draw a diagonal line from top left (0,0) to bottom right (127,63)

```
oled.drawLine(0,0,127,63)
```

Draw a rectangle with no fill:

```
oled.drawRectangle(80,10,120,55,False)
```

To fill the rectangle, change False to True.

Draw a circle with its centre at x,y co-ordinates 35,32 with a radius of 25 pixels and no fill.

```
oled.drawCircle(35,32,25,False)
```

To fill the circle, change False to True. However, the fill does appear a little speckled at the moment.

Drawing images

You can only draw bitmaps on this device. To draw a Raspberry Pi bitmap image in the middle of the screen do the following.

```
oled.drawImage("bitmaps/raspberry-pi-logo.bmp", 35, 0)
```

Don't forget to call the update() function to write the changes to the screen.

Animation

Animation can be achieved in two ways:

- By drawing primitives (lines, circles and rectangles) with changing x,y co-ordinates.
- By displaying a series of bit maps.

In the following example a series of bitmaps called clown1.bmp through clown8.bmp are repeatedly drawn.

```
while True:
    for i in range(1,8):
        image = "bitmaps/clown" + str(i) + ".bmp"
        oled.drawImage(image,35,0)
        oled.update()
        time.sleep(0.02)
```

Drawing a horizontal slider

The following command draws a horizontal slider on line number 8 (Bottom of screen).
The slider part is defined as 0 to 100%. In this example it is 50%.

```
oled.drawHorizontalSlider(50,8)
```

This routine is great for volume controls or display of a particular quantity.

A vertical slider?
That would be a nice project for you.

Programming the IR sensor

The IR sensor is ordered as an optional extra with the cosmic controller board from IQaudIO. See the test_remote.py for an example of the IR routines.

Import the remote IR class.

```
from ir_remote_class import IRRemote
```

Define the listen routine:

```
ir_remote = IRRemote(ir_callback)
ir_remote.listen()
```

Define the call-back routine:

```
def ir_callback(button):
        print "Callback got button:", button
```

If everything is OK then the names of the buttons as shown in Table 2 Remote Control Key names and functions.

The listen routine blocks

The above is very simple to use; however, you cannot just insert the above code into your program if you are using the Cosmic Controller and Oled routines as is. The reason is that the **ir_remote.listen()** routine is blocking. Once called it blocks until a remote-control key is received. This means that it will block all of your other routines. Launching the listen routine as a separate thread does not seem to help either.

You will need to invoke the IR listener routines in either a separate program or process and use inter-process communications between the IR listener and your program. This is advanced programming. In the Bob Rathbone Internet radio software for the Cosmic Controller there is a service (daemon) written for the IR sensor and a small UDP server to do the communications between the IR service and the radio program.

See https://github.com/bobrathbone/piradio6
The relevant source files are:

- remote_control.py
- udp_server_class.py
- rc_daemon.py
- radio_class.py

The following imports must be used:

```
from udp_server_class import UDPServer
from udp_server_class import RequestHandler
```

See the UDPServer routines in the radio_class.py file.

Licences

The software and documentation for this project is released under the GNU General Public Licence.

The GNU General Public License (GNU GPL or GPL) is the most widely used free software license, which guarantees end users (individuals, organizations, companies) the freedoms to use, study, share (copy), and modify the software.

See http://www.gnu.org/licenses for further information on the GNU General Public License.

The licences for the source and documentation for this project are:

GNU General Public License.	See http://www.gnu.org/licenses/gpl.html
GNU AFFERO General Public License.	See http://www.gnu.org/licenses/agpl.html
GNU Free Documentation License.	See http://www.gnu.org/licenses/fdl.html

Disclaimer

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BELIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE OR DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Acknowledgements

Thanks to Olimex Limited for their SSD1306 OLED routines which were used in the **oled_class.py** program and for their excellent technical support. See http://www.olimex.com. Original source https://github.com/SelfDestroyer/pyMOD-OLED.git

Thanks to Gordon Garrity at IQaudIO http://iqaudio.co.uk for his help and sponsorship to develop these routines to support the IQAudIO Cosmic Controller and the SSD1306 OLED display.

The original instructions on how to use Rotary Encoders came from an excellent article by Guy Carpenter. See:
http://guy.carpenter.id.au/gaugette/2013/01/14/rotary-encoder-library-for-the-raspberry-pi/
To Adafruit Industries for their excellent LCD plate and I2C code. See http://www.adafruit.com.

To Sander Marechal for his simple daemon code used in the **cosmicd** code.
Website http://www.jejik.com/articles/2007/02/a_simple_unix_linux_daemon_in_python

To http://www.allaboutcircuits.com for Figure 9 Soldering precautions.

Frequently asked questions (FAQs)

What is the login name and password?

The default login name is: pi
The default password is: raspberry
You should change this at the earliest opportunity.

Is it possible to change the time and date format?

Yes. Please see the **dateformat** parameter in the relevant test program. Seen the date man page for details. Example:

```
dateformat = "%H:%M %d%m"
```

What Rotary Encoder can I use for this project?

Use only the encoder supplied by IQaudIO. Others may not work.

Can this code or documentation be re-used in other projects?

Yes, it can. You can even use it commercially provided that you do so under the terms of the licence distributed with this package. The software and documentation for this project is released under the GNU General Public Licence and may be re-used it other projects. See Licences on page 33.

You do not need to ask permission to re-use this code or documentation as it is already permitted in the licenses.

Technical support

Hardware

Hardware issues should be referred to IQaudIO Limited at:
http://iqaudio.co.uk

Software

Technical support for this software is on a voluntary basis by e-mail only at bob@bobrathbone.com. If there are any problems with this email address then also CC r.h.rathbone@gmail.com. Before asking for support, please first consult the troubleshooting section on page 27. I will always respond to e-mails requesting help and will never ignore them. I only ask that you do the same (i.e. Did my suggestions help or not?). Be sure to provide the following information:

- A clear description of the fault.
- What you have already done to locate the problem?
- Did you run the test programs and what was the result?
- Did you vary from the procedure in the manual or add any other software?
- Please do not answer my questions with a question. Please supply the information requested.

Please note that support for general Raspberry PI problems is not provided. Only issues relating to the Cosmic Controller software will be investigated.

For general Raspberry PI support see the following site:
http://www.raspberrypi.org/forums/

For those of you who want to amend the code to create your own software please note: I am very happy to help people with their projects but my time is limited so I ask that you respect that. Please also appreciate that I cannot engage in long email conversations with every constructor to debug their code or to teach Python.

Glossary

HDMI	High-Definition Multimedia Interface for audio and video plus Ethernet interface.
IP	Internet Protocol – IP address
IR	Infra Red – In this case for the IR remote control
LED	Light Emitting Diode
OLED	Organic Light Emitting Diode
SD	San Disk Memory Card commonly found in cameras and Smartphone's
SSH	Secure Shell – Encrypted terminal
TCP/IP	The common name for network protocols used by the Internet and computer networks.

Appendix A

Appendix A.1 wiring

The following table shows the default wiring scheme for the Cosmic Controller when plugged in directly to the Raspberry Pi. The design allows other GPIOs to be used via the 15-pin single in line connector. In such a case the configuration in the **config_class.py** source file must also be changed.

Table 3 Radio and DAC devices 40 pin wiring

Pin	Description	Controller Function	Name	Audio DAC Function	OLED Connector Pin
1	3V3	+3V supply	+3V	+3V	
2	5V	5V for OLED	+5V	+5V	
3	GPIO2	I2C Data	I2C Data	I2C Data	
4	5V			+5V	
5	GPIO3	I2C Clock	I2C Clock	I2C Clock	
6	GND	Zero volts	0V	0V	
7	GPIO 4	Left push button			
8	GPIO 14	LED 1	UART TX		
9	GND	Zero Volts		0V	
10	GPIO 15	LED 2	UART RX		
11	GPIO 17	Menu switch			
12	GPIO 18			I2S CLK	
13	GPIO 27				
14	GND	Zero Volts		0V	
15	GPIO 22			Mute	
16	GPIO 23	Encoder output A		Rotary enc A	
17	3V3	+3V supply		0V	
18	GPIO 24	Encoder output B		Rotary Enc B	
19	GPIO 10		SPI-MOSI		
20	GND	Zero Volts			
21	GPIO9		SPI-MISO		
22	GPIO 25	IR Sensor in		IR sensor	
23	GPIO 11		SPI-SCLK		
24	GPIO 8	Unused	SPI-CE0		
25	GND	Zero Volts		0V	
26	GPIO 7	Unused	SPI-CE1		
27	DNC			PiDac+ Eprom	
28	DNC			PiDac+ Eprom	
29	GPIO5	Middle push-button			
30	GND	Zero Volts			
31	GPIO6	Right push-button			
32	GPIO12	Unused			
33	GPIO 13	Unused			

34	GND	Zero Volts		
35	GPIO 19	IQaudIO DAC+	I2S	I2S
36	GPIO 16	LED 3 (IR LED out)		
37	GPIO 26			
38	GPIO 20	IQaudIO DAC+	I2S DIN	I2S DIN
39	GND	Zero Volts		
40	GPIO 21	IQaudIO DAC+	I2S DOUT	I2S DOUT

Colour Legend	Controller	IQaudIO DAC	I2C (shared)	SPI Interface

Appendix A.2 The cosmicd.service file

The **cosmicd.service** file is responsible for how the **cosmicd** service is started by **systemd**.

```
# IQAudio cosmic controller systemd script
# $Id: cosmicd.service,v 1.1 2018/05/16 11:27:01 bob Exp $
[Unit]
Description=Cosmic controller daemon
After=network.target

[Service]
Type=simple
ExecStart=/usr/share/cosmicd/cosmicd.py nodaemon

[Install]
WantedBy=multi-user.target
```

Note: If creating your own service file, you can safely remove the **$Id:** line.

Appendix A.4 Technical details

Below are the technical details of the hardware used in this project.

Raspberry Pi: Model 3B, Revision 1.2, RAM: 1024 MB, Maker: Sony, Onboard WiFi and Bluetooth 4.1

Operating system: Tested on Raspbian GNU/Linux 9 (Stretch)

OLED Display: Olimex SSD1306 128 by 64-pixel OLED display

User interface: IQAudio Cosmic Controller with three buttons and rotary encoder and three LEDs. See http://iqaudio.co.uk/

Web site: http://www.bobrathbone.com

Appendix A.4 Source files

Test programs

test_controller.py	Test rotary encoder, buttons and LEDs
test_oled.py	Test basic text functions on the OLED
test_graphics.py	Test graphics functions
test_remote.py	Test the remote-control IR sensor
cosmicd.py	A test daemon for the Cosmic controller and OLED

Classes

config_class.py	Default configuration (Amend if necessary)
cosmic_class.py	Cosmic controller functions class
oled_class.py	OLED functions class
ir_remote_class.py	IR sensor class
status_led_class.py	Cosmic controller status LEDs class

Olimex source

oled/Font.py	Font and text routines
oled/Graphics.py	Graphics primitives
oled/OLED.py	I2C interface to OLED
oled/__init__.py	Init plus version

Bit maps

See bitmaps/*.bmp

Package build

build.sh	Package build script
cosmicd	Package build definition
cosmicd.preinst	Package pre-installation routines skeleton
cosmicd.postinst	Package post-installation routines skeleton
cosmicd.postrm	Package remove routines skeleton

The package will install the source in **/usr/share/cosmicd/** directory

Index