**Yocto**

**Ashok L:**

## Building Raspberry Pi Systems with Yocto

15 Apr 2017

Building systems for [Raspberry Pi](https://www.raspberrypi.org/) boards using tools from the [Yocto Project](https://www.yoctoproject.org/).

The example images in [meta-rpi](https://github.com/jumpnow/meta-rpi) build systems that support C, C++, [Qt5](http://www.qt.io/), Perl and Python development, the languages and tools that I commonly use. Other languages are supported, but you will have to add the packages to your image recipe.

Yocto is a good tool for building minimal, customized systems like one for a dedicated hacking project or more commonly for industrial or commercial embedded products.

If you are looking for a full-featured desktop experience you will probably be better off sticking with [Raspbian](https://www.raspbian.org/) or another of the more popular, user friendly [RPi distributions](https://www.raspberrypi.org/downloads/).

If the following are important to you then a build system like this (Yocto) or [Buildroot](http://www.jumpnowtek.com/rpi/Raspberry-Pi-Systems-with-Buildroot.html) might be what you are looking for.

* Small image sizes
* Quick boot times
* Read-only rootfs
* Tight control of all software installed

You can build systemd systems with Yocto, but the default images I’m building from [meta-rpi](https://github.com/jumpnow/meta-rpi) use sysvinit.

If you are [Qt5](http://www.qt.io/) developer then you will appreciate that the RPi comes with working OpenGL drivers for the GPU. This means [Qt OpenGL](http://doc.qt.io/qt-5/qtopengl-index.html) and [Qt QuickControls2](http://doc.qt.io/qt-5/qtquickcontrols2-index.html) applications will work when using the [eglfs](http://doc.qt.io/qt-5/embedded-linux.html) platform plugin.

Here is another post with more details on [developing with Qt5 on the RPi](http://www.jumpnowtek.com/rpi/Qt5-and-QML-Development-with-the-Raspberry-Pi.html).

I am using the Yocto [meta-raspberrypi](http://git.yoctoproject.org/cgit/cgit.cgi/meta-raspberrypi) layer, but have updated recipes for the Linux kernel, [gpu firmware](https://github.com/raspberrypi/firmware) and some [userland](https://github.com/raspberrypi/userland) components.

I’ve done some testing with the following boards

* [RPi3](https://www.raspberrypi.org/products/raspberry-pi-3-model-b/)
* [RPi2](https://www.raspberrypi.org/products/raspberry-pi-2-model-b/) including the Model B v1.2
* [RPi Zero](https://www.raspberrypi.org/products/pi-zero/)
* [RPi Zero W](https://www.raspberrypi.org/products/pi-zero-w/)
* [RPi 1 Model B](https://www.raspberrypi.org/products/model-b/)
* [RPi compute module](https://www.raspberrypi.org/products/compute-module/) with the [Raspberry Pi Compute Module Dev Kit](https://www.raspberrypi.org/products/compute-module-development-kit/)
* [RPi compute module](https://www.raspberrypi.org/products/compute-module/) with the [Gumstix Pi Compute Dev Board](https://store.gumstix.com/expansion/partners-3rd-party/gumstix-pi-compute-dev-board.html)
* [RPi compute module](https://www.raspberrypi.org/products/compute-module/) with the [Western Digital Media Stick](http://store.wdc.com/store/wdus/en_US/DisplayAccesoryProductDetailsPage/ThemeID.40718400/Accessories/Media_Stick_for_Raspberry_Pi/productID.331153900/categoryId.70262300)

All boot fine. Ethernet works where applicable. HDMI and USB work. RPi3 wifi works, I have not tried the RPi3 bluetooth. I have it disabled so I can use the serial console.

The serial console works off the header pins on all the boards.

SPI, I2C and generic GPIO are all standard embedded Linux stuff. DTS overlays are available for common configurations.

I have one RPi2 running as my office [music system](http://www.jumpnowtek.com/rpi/Raspberry-Pi-Pandora-music-player.html).

I also frequently use RPis as my Linux test platform for Qt applications. I do most Qt development on a workstation, usually Windows, but eventually most applications have to run on Linux and MacOS as well. The RPi3s work great both for compiling and running Qt5 applications.

### Downloads

If you want a quick look at the resulting systems, you can download an example for the RPi 2/3 image [here](http://www.jumpnowtek.com/downloads/rpi/).

Instructions for installing onto an SD card are in the [README](http://www.jumpnowtek.com/downloads/rpi/README.txt).

### System Info

The Yocto version is 2.2.1 the [morty] branch.

The 4.9.22 Linux kernel comes from the [github.com/raspberrypi/linux](https://github.com/raspberrypi/linux) repository.

These are sysvinit systems using [eudev](https://wiki.gentoo.org/wiki/Project:Eudev).

The Qt version is 5.7.1 There is no X11 and no desktop installed. [Qt](http://www.qt.io/) GUI applications can be run fullscreen using one of the [Qt embedded linux plugins](http://doc.qt.io/qt-5/embedded-linux.html) like eglfs or linuxfb which are both provided.

Perl 5.22.1, Python 2.7.12 and Python3 3.5.2 each with a number of modules is included.

[omxplayer](http://elinux.org/Omxplayer) for playing video and audio files from the command line, hardware accelerated.

[Raspicam](https://www.raspberrypi.org/documentation/usage/camera/raspicam/README.md) command line tools for using the Raspberry Pi camera module.

An example Raspberry Pi [music system](http://www.jumpnowtek.com/rpi/Raspberry-Pi-Pandora-music-player.html) using an [IQaudIO Pi-DigiAMP+](http://www.iqaudio.co.uk/home/9-pi-digiamp-0712411999650.html) add-on board and [pianobar](https://6xq.net/pianobar/), a console-based client for [Pandora](http://www.pandora.com/) internet radio.

That system also works with the [HiFiBerry Amp+](https://www.hifiberry.com/ampplus/) board.

The Adafruit [PiTFT 3.5”](https://www.adafruit.com/products/2441) and [PiTFT 2.8”](https://www.adafruit.com/products/1601) resistive touchscreens work. Support for some other TFT displays is included, but I haven’t tested them.

[Raspi2fb](https://github.com/AndrewFromMelbourne/raspi2fb) is included for mirroring the GPU framebuffer to the small TFT displays. This allows for running Qt GUI applications on the TFTs.

As of 2017-04-15, here is the list of device tree overlays installed

root@rpi3:~# uname -a

Linux rpi3 4.9.22 #1 SMP Sat Apr 15 08:37:58 EDT 2017 armv7l armv7l armv7l GNU/Linux

root@rpi3:~# ls /mnt/fat/overlays/

adau1977-adc.dtbo mmc.dtbo

adau7002-simple.dtbo mz61581.dtbo

ads1015.dtbo pi3-act-led.dtbo

ads1115.dtbo pi3-disable-bt.dtbo

ads7846.dtbo pi3-disable-wifi.dtbo

akkordion-iqdacplus.dtbo pi3-miniuart-bt.dtbo

allo-boss-dac-pcm512x-audio.dtbo piscreen.dtbo

allo-piano-dac-pcm512x-audio.dtbo piscreen2r.dtbo

allo-piano-dac-plus-pcm512x-audio.dtbo pisound.dtbo

at86rf233.dtbo pitft22.dtbo

audioinjector-addons.dtbo pitft28-capacitive.dtbo

audioinjector-wm8731-audio.dtbo pitft28-resistive.dtbo

audremap.dtbo pitft35-resistive.dtbo

bmp085\_i2c-sensor.dtbo pps-gpio.dtbo

dht11.dtbo pwm-2chan.dtbo

dionaudio-loco-v2.dtbo pwm.dtbo

dionaudio-loco.dtbo qca7000.dtbo

dpi18.dtbo raspidac3.dtbo

dpi24.dtbo rpi-backlight.dtbo

dwc-otg.dtbo rpi-cirrus-wm5102.dtbo

dwc2.dtbo rpi-dac.dtbo

enc28j60-spi2.dtbo rpi-display.dtbo

enc28j60.dtbo rpi-ft5406.dtbo

fe-pi-audio.dtbo rpi-proto.dtbo

googlevoicehat-soundcard.dtbo rpi-sense.dtbo

gpio-ir.dtbo rpi-tv.dtbo

gpio-poweroff.dtbo rra-digidac1-wm8741-audio.dtbo

hifiberry-amp.dtbo sc16is750-i2c.dtbo

hifiberry-dac.dtbo sc16is752-spi1.dtbo

hifiberry-dacplus.dtbo sdhost.dtbo

hifiberry-digi-pro.dtbo sdio-1bit.dtbo

hifiberry-digi.dtbo sdio.dtbo

hy28a.dtbo sdtweak.dtbo

hy28b.dtbo smi-dev.dtbo

i2c-bcm2708.dtbo smi-nand.dtbo

i2c-gpio.dtbo smi.dtbo

i2c-mux.dtbo spi-gpio35-39.dtbo

i2c-pwm-pca9685a.dtbo spi-rtc.dtbo

i2c-rtc.dtbo spi0-cs.dtbo

i2c-sensor.dtbo spi0-hw-cs.dtbo

i2c0-bcm2708.dtbo spi1-1cs.dtbo

i2c1-bcm2708.dtbo spi1-2cs.dtbo

i2s-gpio28-31.dtbo spi1-3cs.dtbo

iqaudio-dac.dtbo spi2-1cs.dtbo

iqaudio-dacplus.dtbo spi2-2cs.dtbo

iqaudio-digi-wm8804-audio.dtbo spi2-3cs.dtbo

justboom-dac.dtbo tinylcd35.dtbo

justboom-digi.dtbo uart1.dtbo

lirc-rpi.dtbo vc4-fkms-v3d.dtbo

mcp23017.dtbo vc4-kms-v3d.dtbo

mcp23s17.dtbo vga666.dtbo

mcp2515-can0.dtbo w1-gpio-pullup.dtbo

mcp2515-can1.dtbo w1-gpio.dtbo

mcp3008.dtbo wittypi.dtbo

midi-uart0.dtbo

I’ve tested a few

* ads1015
* hifiberry-amp
* i2c-rtc
* iqaudio-dacplus
* mcp2515-can0
* mcp3008
* pi3-disable-bt
* pitft28-resistive
* pitft35-resistive
* sdhost (the default, but you can overclock now)
* spi1-1cs
* spi1-2cs

They all come from the official Raspberry Pi kernel tree so I have confidence they all work fine. I need more hardware to test many of them.

### Ubuntu Setup

I am primarily using 16.04 64-bit servers for builds. Older versions should work.

You will need at least the following packages installed

build-essential

chrpath

diffstat

gawk

git

libncurses5-dev

pkg-config

subversion

texi2html

texinfo

For 16.04 you also need to install the python 2.7 package that the Yocto 2.2 branch requires

python2.7

And then create a link for it in /usr/bin

sudo ln -sf /usr/bin/python2.7 /usr/bin/python For all versions of Ubuntu, you should change the default Ubuntu shell from `dash` to `bash` by running this command from a shell

sudo dpkg-reconfigure dash

Choose No to dash when prompted.

### Fedora Setup

I have used a Fedora 25 64-bit workstation.

The extra packages I needed to install for Yocto were

chrpath

perl-bignum

perl-Thread-Queue

texinfo

and the package group

Development Tools

There might be more packages required since I had already installed qt-creator and the Development Tools group before I did the first build with Yocto.

Fedora already uses bash as the shell.

### Clone the dependency repositories

First the main Yocto project poky repository, use the [morty] branch

scott@octo:~ git clone -b morty git://git.yoctoproject.org/poky.git poky-morty

The meta-openembedded repository, use the [morty] branch

scott@octo:~$ cd poky-morty

scott@octo:~/poky-morty$ git clone -b morty git://git.openembedded.org/meta-openembedded

The meta-qt5 repository, use the [morty] branch

scott@octo:~/poky-morty$ git clone -b morty https://github.com/meta-qt5/meta-qt5.git

And finally the meta-raspberrypi repository. use the [morty] branch

scott@octo:~/poky-morty$ git clone -b morty git://git.yoctoproject.org/meta-raspberrypi

Those 4 repositories shouldn’t need modifications other then updates and can be reused for different projects or different boards.

### Clone the meta-rpi repository

Create a separate sub-directory for the meta-rpi repository before cloning. This is where you will be doing your customization.

scott@octo:~$ mkdir ~/rpi

scott@octo:~$ cd ~/rpi

scott@octo:~/rpi$ git clone -b morty git://github.com/jumpnow/meta-rpi

The meta-rpi/README.md file has the last commits from the dependency repositories that I tested. You can always checkout those commits explicitly if you run into problems.

### Initialize the build directory

Much of the following are only the conventions that I use. All of the paths to the meta-layers are configurable.

Choose a build directory. I tend to do this on a per board and/or per project basis so I can quickly switch between projects. For this example I’ll put the build directory under ~/rpi/ with the meta-rpi layer.

You could manually create the directory structure like this

scott@octo:~$ mkdir -p ~/rpi/build/conf

Or you could use the Yocto environment script oe-init-build-env like this passing in the path to the build directory

scott@octo:~$ **source poky-morty/oe-init-build-env ~/rpi/build**

The Yocto environment script will create the build directory if it does not already exist.

### Customize the configuration files

There are some sample configuration files in the meta-rpi/conf directory.

Copy them to the build/conf directory (removing the ‘-sample’)

scott@octo:~/rpi$ cp meta-rpi/conf/local.conf.sample build/conf/local.conf

scott@octo:~/rpi$ cp meta-rpi/conf/bblayers.conf.sample build/conf/bblayers.conf

If you used the oe-init-build-env script to create the build directory, it generated some generic configuration files in the build/conf directory. It is okay to copy over them.

It is not necessary, but you may want to customize the configuration files before your first build.

Do not use the ‘~’ character when defining directory paths in the configuration files.

### Edit bblayers.conf

In bblayers.conf file replace ${HOME} with the appropriate path to the meta-layer repositories on your system if you modified any of the paths in the previous instructions.

NOTE: Do not include meta-yocto-bsp in your bblayers.conf. The BSP requirements for the BBB are included in meta-bbb.

For example, if your directory structure does not look exactly like this, you will need to modify bblayers.conf

~/poky-morty/

meta-openembedded/

meta-qt5/

meta-raspberrypi

...

~/rpi/

meta-rpi/

build/

conf/

### Edit local.conf

The variables you may want to customize are the following:

* MACHINE
* TMPDIR
* DL\_DIR
* SSTATE\_DIR

The defaults for all of these work fine. Adjustments are optional.

##### MACHINE

The choices are raspberrypi2 the default or raspberrypi.

Use raspberrypi2 for the RPi2, RPi3 or CM3.

Use raspberry for the RPi0 or original CM.

There is a new raspberrypi3 MACHINE option with [morty], but I recommend you stick with using raspberrypi2 for MACHINE. Nothing is lost.

You can only build for one type of MACHINE at a time because of the different instruction sets.

##### TMPDIR

This is where temporary build files and the final build binaries will end up. Expect to use at least 50GB. You probably want at least 80GB available.

The default location is in the build directory, in this example ~/rpi/build/tmp.

If you specify an alternate location as I do in the example conf file make sure the directory is writable by the user running the build.

##### DL\_DIR

This is where the downloaded source files will be stored. You can share this among configurations and builds so I always create a general location for this outside the project directory. Make sure the build user has write permission to the directory you decide on.

The default location is in the build directory, ~/rpi/build/sources.

##### SSTATE\_DIR

This is another Yocto build directory that can get pretty big, greater then 8GB. I often put this somewhere else other then my home directory as well.

The default location is in the build directory, ~/rpi/build/sstate-cache.

### Run the build

You need to [source](http://stackoverflow.com/questions/4779756/what-is-the-difference-between-source-script-sh-and-script-sh) the Yocto environment into your shell before you can use [bitbake](http://www.yoctoproject.org/docs/2.1/bitbake-user-manual/bitbake-user-manual.html). The oe-init-build-env will not overwrite your customized conf files.

scott@octo:~$ source poky-morty/oe-init-build-env ~/rpi/build

### Shell environment set up for builds. ###

You can now run 'bitbake '

Common targets are:

core-image-minimal

core-image-sato

meta-toolchain

meta-toolchain-sdk

adt-installer

meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'

scott@octo:~/rpi/build$

I don’t use those Common targets, but instead use my own custom image recipes.

There are three example images available in the meta-rpi layer. The recipes for the images can be found in meta-rpi/images/

* console-image.bb
* qt5-basic-image.bb
* qt5-image.bb
* audio-image.bb
* iot-image.bb

You should add your own custom images to this same directory.

#### console-image

A basic console developer image. See the recipe meta-rpi/images/console-image.bb for specifics, but some of the installed programs are

gcc/g++ and associated build tools

git

ssh/scp server and client

perl and python with a number of modules

pi-blaster

omxplayer

raspicam utilities

The console-image has a line

inherit core-image

which is poky-morty/meta/classes/core-image.bbclass and pulls in some required base packages. This is useful to know if you create your own image recipe.

#### qt5-basic-image

This image includes the console-image and adds Qt5 with the associated development headers and qmake. This packages included in this image are sufficient to develop basic QWidgets apps and typically what I use.

#### qt5-image

Adds to the qt5-basic-image the following Qt packages with libs, header files and mkspecs

* qt3d
* qtcharts
* qtconnectivity
* qtdeclarative
* qtgraphicaleffects
* qtlocation
* qtmultimedia
* qtquickcontrols2
* qtsensors
* qtserialbus
* qtsvg
* qtwebsockets
* qtvirtualkeyboard
* qtxmlpatterns

I am not normally a QML or Qt OpenGL developer, but I did test a number of the [Qt Examples](http://doc.qt.io/qt-5/qtexamplesandtutorials.html) and all that I tried compiled and worked.

#### audio-image

See this [post](http://www.jumpnowtek.com/rpi/Raspberry-Pi-Pandora-music-player.html) for details on using this image.

#### iot-image

Adds [mosquitto](http://mosquitto.org/), the [paho python client](https://eclipse.org/paho/clients/python/) and the [python-flask](http://flask.pocoo.org/) packages to the console-image.

### Build

To build the console-image run the following command

scott@octo:~/rpi/build$ bitbake console-image

You may occasionally run into build errors related to packages that either failed to download or sometimes out of order builds. The easy solution is to clean the failed package and rerun the build again.

For instance if the build for zip failed for some reason, I would run this

scott@octo:~/rpi/build$ bitbake -c cleansstate zip

scott@octo:~/rpi/build$ bitbake zip

And then continue with the full build.

scott@octo:~/rpi/build$ bitbake console-image

To build the qt5-image it would be

scott@octo:~/rpi/build$ bitbake qt5-image

The cleansstate command (with two s’s) works for image recipes as well.

The image files won’t get deleted from the TMPDIR until the next time you build.

### Copying the binaries to an SD card (or eMMC)

After the build completes, the bootloader, kernel and rootfs image files can be found in <TMPDIR>/deploy/images/raspberrypi2/ or <TMPDIR>/deploy/images/raspberrypi depending on MACHINE.

The meta-rpi/scripts directory has some helper scripts to format and copy the files to a microSD card.

See [this post](http://www.jumpnowtek.com/rpi/Working-with-the-raspberry-pi-compute.html) for an additional first step required for the [RPi Compute](https://www.raspberrypi.org/products/compute-module/) eMMC.

#### mk2parts.sh

This script will partition an SD card with the minimal 2 partitions required for the RPI.

Insert the microSD into your workstation and note where it shows up.

[lsblk](http://linux.die.net/man/8/lsblk) is convenient for finding the microSD card.

For example

scott@octo:~/rpi/meta-rpi$ lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sda 8:0 0 931.5G 0 disk

|-sda1 8:1 0 93.1G 0 part /

|-sda2 8:2 0 93.1G 0 part /home

|-sda3 8:3 0 29.8G 0 part [SWAP]

|-sda4 8:4 0 1K 0 part

|-sda5 8:5 0 100G 0 part /oe5

|-sda6 8:6 0 100G 0 part /oe6

|-sda7 8:7 0 100G 0 part /oe7

|-sda8 8:8 0 100G 0 part /oe8

|-sda9 8:9 0 100G 0 part /oe9

`-sda10 8:10 0 215.5G 0 part /oe10

sdb 8:16 1 7.4G 0 disk

|-sdb1 8:17 1 64M 0 part

`-sdb2 8:18 1 7.3G 0 part

I would use sdb for the format and copy script parameters on this machine.

It doesn’t matter if some partitions from the SD card are mounted. The mk2parts.sh script will unmount them.

BE CAREFUL with this script. It will format any disk on your workstation.

scott@octo:~$ cd ~/rpi/meta-rpi/scripts

scott@octo:~/rpi/meta-rpi/scripts$ sudo ./mk2parts.sh sdb

You only have to format the SD card once.

#### /media/card

You will need to create a mount point on your workstation for the copy scripts to use.

scott@octo:~$ sudo mkdir /media/card

You only have to create this directory once.

#### copy\_boot.sh

This script copies the BCM2835 bootloader files, the Linux kernel, dtbs for both RPi 2 and RPi boards and a number of DTB overlays (that I have not tried) to the boot partition of the SD card.

This copy\_boot.sh script needs to know the TMPDIR to find the binaries. It looks for an environment variable called OETMP.

For instance, if I had this in the local.conf

TMPDIR = "/oe8/rpi/tmp-morty"

Then I would export this environment variable before running copy\_boot.sh

scott@octo:~/rpi/meta-rpi/scripts$ export OETMP=/oe8/rpi/tmp-morty

If you didn’t override the default TMPDIR in local.conf, then set it to the default TMPDIR

scott@octo:~/rpi/meta-rpi/scripts$ export OETMP=~/rpi/build/tmp

The copy\_boot.sh script also needs a MACHINE environment variable specifying the type of RPi board.

scott@octo:~/rpi/meta-rpi/scripts$ export MACHINE=raspberrypi2

or

scott@octo:~/rpi/meta-rpi/scripts$ export MACHINE=raspberrypi

Then run the copy\_boot.sh script passing the location of SD card

scott@octo:~/rpi/meta-rpi/scripts$ ./copy\_boot.sh sdb

This script should run very fast.

#### copy\_rootfs.sh

This script copies the root file system to the second partition of the SD card.

The copy\_rootfs.sh script needs the same OETMP and MACHINE environment variables.

The script accepts an optional command line argument for the image type, for example console or qt5-x11. The default is console if no argument is provided.

The script also accepts a hostname argument if you want the host name to be something other then the default raspberrypi2.

Here’s an example of how you’d run copy\_rootfs.sh

scott@octo:~/rpi/meta-rpi/scripts$ ./copy\_rootfs.sh sdb console

or

scott@octo:~/rpi/meta-rpi/scripts$ ./copy\_rootfs.sh sdb qt5-x11 rpi2

The copy\_rootfs.sh script will take longer to run and depends a lot on the quality of your SD card. With a good Class 10 card it should take less then 30 seconds.

The copy scripts will NOT unmount partitions automatically. If an SD card partition is already mounted, the script will complain and abort. This is for safety, mine mostly, since I run these scripts many times a day on different machines and the SD cards show up in different places.

Here’s a realistic example session where I want to copy already built images to a second SD card that I just inserted.

scott@octo:~$ sudo umount /dev/sdb1

scott@octo:~$ sudo umount /dev/sdb2

scott@octo:~$ export OETMP=/oe8/rpi/tmp-morty

scott@octo:~$ export MACHINE=raspberrypi2

scott@octo:~$ cd rpi/meta-rpi/scripts

scott@octo:~/rpi/meta-rpi/scripts$ ./copy\_boot.sh sdb

scott@octo:~/rpi/meta-rpi/scripts$ ./copy\_rootfs.sh sdb console rpi

Both copy\_boot.sh and copy\_rootfs.sh are simple scripts easily modified for custom use. Once I get past the development stage I usually wrap them both with another script for convenience.

#### Some custom package examples

[spiloop](https://github.com/scottellis/spiloop) is a spidev test application installed in /usr/bin.

The bitbake recipe that builds and packages spiloop is here

meta-rpi/recipes-misc/spiloop/spiloop\_1.0.bb

Use it to test the spidev driver before and after placing a jumper between pins 19 and 21.

[tspress](https://github.com/scottellis/tspress) is a Qt5 GUI application installed in /usr/bin with the qt5-image. I use it for testing touchscreens.

The bitbake recipe is here and can be used a guide for your own applications.

meta-rpi/recipes-qt/tspress/tspress.bb

Check the README in the [tspress](https://github.com/scottellis/tspress) repository for usage.

#### Adding additional packages

To display the list of available packages from the meta- repositories included in bblayers.conf

scott@octo:~$ source poky-morty/oe-init-build-env ~/rpi/build

scott@octo:~/rpi/build$ bitbake -s

Once you have the package name, you can choose to either

1. Add the new package to the console-image or qt5-image, whichever you are using.
2. Create a new image file and either include the console-image the way the qt5-image does or create a complete new image recipe. The console-image can be used as a template.

The new package needs to get included directly in the IMAGE\_INSTALL variable or indirectly through another variable in the image file.

#### Playing videos

The RPi project has a hardware-accelerated, command-line video player called [omxplayer](http://elinux.org/Omxplayer).

Here’s a reasonably sized example from the [Blender](https://www.blender.org/) project to test

root@rpi3:~# wget https://download.blender.org/demo/movies/Cycles\_Demoreel\_2015.mov

You can play it like this (-o hdmi for hdmi audio)

root@rpi3:~# omxplayer -o hdmi Cycles\_Demoreel\_2015.mov

Video codec omx-h264 width 1920 height 1080 profile 77 fps 25.000000

Audio codec aac channels 2 samplerate 48000 bitspersample 16

Subtitle count: 0, state: off, index: 1, delay: 0

V:PortSettingsChanged: 1920x1080@25.00 interlace:0 deinterlace:0 anaglyph:0 par:1.25 display:0 layer:0 alpha:255 aspectMode:0

If you get errors like this

COMXAudio::Decode timeout

Increase memory allocated to the GPU in config.txt

gpu\_mem=128

The RPi GPU can support more then one display, (the DSI display is the default), though apps have to be built specifically to support the second display. Omxplayer is an app with this ability.

So for example, with the RPi DSI touchscreen and an HDMI display attached at the same time, you could run a video on the HDMI display from the touchscreen this way

root@rpi3:~# omxplayer --display=5 -o hdmi Cycles\_Demoreel\_2015.mov

Video codec omx-h264 width 1920 height 1080 profile 77 fps 25.000000

Audio codec aac channels 2 samplerate 48000 bitspersample 16

Subtitle count: 0, state: off, index: 1, delay: 0

V:PortSettingsChanged: 1920x1080@25.00 interlace:0 deinterlace:0 anaglyph:0 par:1.25 display:5 layer:0 alpha:255 aspectMode:0

I was not able to run a eglfs Qt app on the RPi DSI display while playing a movie with omxplayer on the HDMI display. Perhaps a linuxfb Qt app that doesn’t use the GPU could run simultaneously. Some more testing is needed.

#### Using the Raspberry Pi Camera

The [raspicam](https://www.raspberrypi.org/documentation/raspbian/applications/camera.md) command line tools are installed with the console-image.

* raspistill
* raspivid
* raspiyuv

To enable the RPi camera, add or edit the following in the RPi configuration file config.txt

start\_x=1

gpu\_mem=128

disable\_camera\_led=1 # optional for disabling the red LED on the camera

To get access to config.txt, mount the boot partition first

root@rpi# mkdir /mnt/fat

root@rpi# mount /dev/mmcblk0p1 /mnt/fat

Then edit, save and reboot.

root@rpi# vi /mnt/fat/config.txt

or

root@rpi# nano /mnt/fat/config.txt

A quick test of the camera, flipping the image because of the way I have my camera mounted and a timeout of zero so it runs until stopped.

root@rpi2# raspistill -t 0 -hf -vf

#### PWM

There are [two hardware timers](http://www.jumpnowtek.com/rpi/Using-the-Raspberry-Pi-Hardware-PWM-timers.html) with kernel support available on the RPi’s with 40 pin headers. Only one of the timers is available on the original RPi 1 with it’s 26 pin header.

The console-image contains a utility called [pi-blaster](http://www.jumpnowtek.com/rpi/Working-with-pi-blaster-on-the-RPi.html) that can be used to efficiently drive PWM outputs from gpio pins.