RaspberryPi Emulation Using

QEMU

ELECTGON
www.electgon.com
ma_ext@gmx.net

15.04.2018

Contents

Abstract

This tutorial was set as an illustration about how to use QEMU for emulation of ARM development boards, namely Raspberry pi board. It will start with quick demonstration about how using QEMU for virtualization. Then detailed guide will be shown to learn how to build emulation environment for Raspberry in QEMU.

1 Introduction

Developing a project for an ARM board requires testing of your project performance on this target board. In case of target board is not available to work with, you can rely on emulation techniques to run your environment. QEMU can be used for that purpose. Generally QEMU is used for virtualization and Emulation. In the following lines, needed procedure for setting and running QEMU as emulation for ARM development board will be discussed.

2 QEMU Installation

You can install QEMU directly by typing

```
$sudo apt-get install qemu
```

or by getting source code from

```
$git clone git://git.qemu-project.org/qemu.git
```

then configure and install. It is preferred to use latest method, in order to configure the installation.

Before start installation, some packages should be installed on your machine first,

```
$sudo apt-get install zlib1g_dev
$sudo apt-get install libglib2.0-dev
$sudo apt-get install libpixman-1-dev
$sudo apt-get install libfdt
$sudo apt-get install gvncviewer
$sudo apt-get install libsdl1.2-dev
```

Now you can start QEMU installation using the following steps:

- Create directory to clone source code in it

```
$mkdir WORKING_DIR
$cd WORKING_DIR
$git clone git://git.qemu-project.org/qemu.git
$cd qemu
$./configure --target-list=x86_64-softmmu --enable-debug --enable-sdl
$make
$make install
```

Note that in this line "./configure –target-list=x86_64-softmmu –enable-debug –enable-sdl" we configured QEMU to support x86 processors, this is in case we need to virtualize or emulate PC machine. But later we will need to emulate ARM processor so it is recommended to run this line as

```
$./configure --target-list"=x86_64-softmmu arm-softmmu arm-linux-"user --enable
    -debug --enable-sdl
```

3 Running QEMU

In general to run QEMU you need to provide the operating system that will run and disk space for running it. You can imagine any qemu command should have

```
$qemu [Running System Image] [Disk Image]
```

Some other attributes are needed sometimes, but this will be discussed later.

You need then to have an image of your disk “Hardware part” and an ISO image of the operating system that you need to virtualize “Software part”. Now lets see how to work with QEMU as a virtualizer to get familiar with QEMU important instructions. Then we can see how to work with QEMU as ARM board emulator.

To get a running system image, In this tutorial I downloaded Ubuntu iso file from Ubuntu website, my downloaded version is “ubuntu-14.04.2-desktop-amd64”.

We need now to create disk image that will host this operating system. lets work in WORKING_DIR

```
$cd path/to/WORKING_DIR
$qemu-img create -f qcow tutorial_disk.img 3G
$qemu-system-x86_64 -cdrom ubuntu-14.04.2-desktop-amd64.iso -hda \
 tutorial_disk.img -m 1024 -boot d
```

In the command “qemu-img create -f qcow tutorial_disk.img 3G” we created an image of the harddisk with size 3G.

In the last command “-cdrom” is used to indicate for qemu as this iso file is running as from cdrom. “-hda” to direct qemu that this system will run on a harddisk “tutorial_disk.img”. “-m” is used to indicate what the RAM speed which is in our example 1024 M.

your operating system should run then. Note, if you found this message at the start “Failed to access perfctr msr” this is not error message, it says that the CPU doesn’t support performance counters. You should have your system as shown in figure 1. You can continue with setup of your operating system.

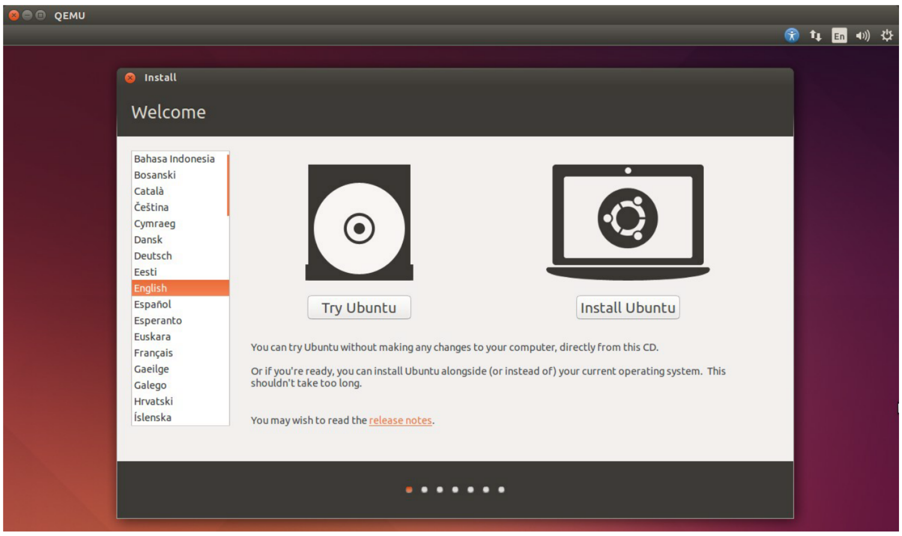


Figure 1: Ubuntu Running in QEMU

4 QEMU for ARM Development Boards

To evaluate a board, specification of this board should be provided to QEMU. These specifications are built in QEMU library files. So when you configure QEMU, you actually choose which library files should be compiled during installation . For example we configured QEMU to support x86 and ARM using the command

```
$./configure --target-list”=x86_64-softmmu arm-softmmu arm-linux-”user
```

Since ARM has different models and versions of the processor, we need to check of our target model is support in QEMU or not. You can know that by typing *qemu-arm -cpu ?* You should get a list of available supported ARM processors.

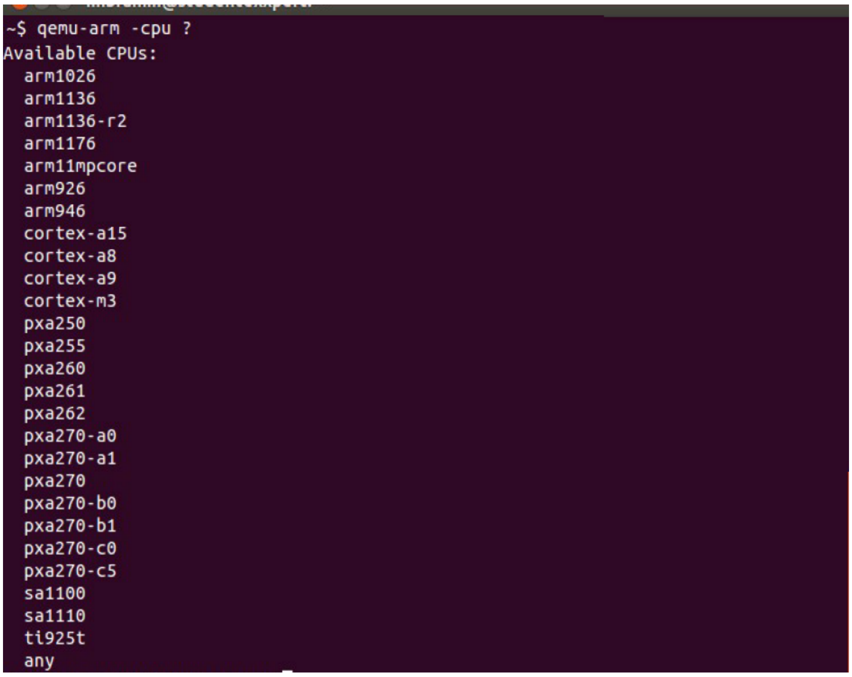


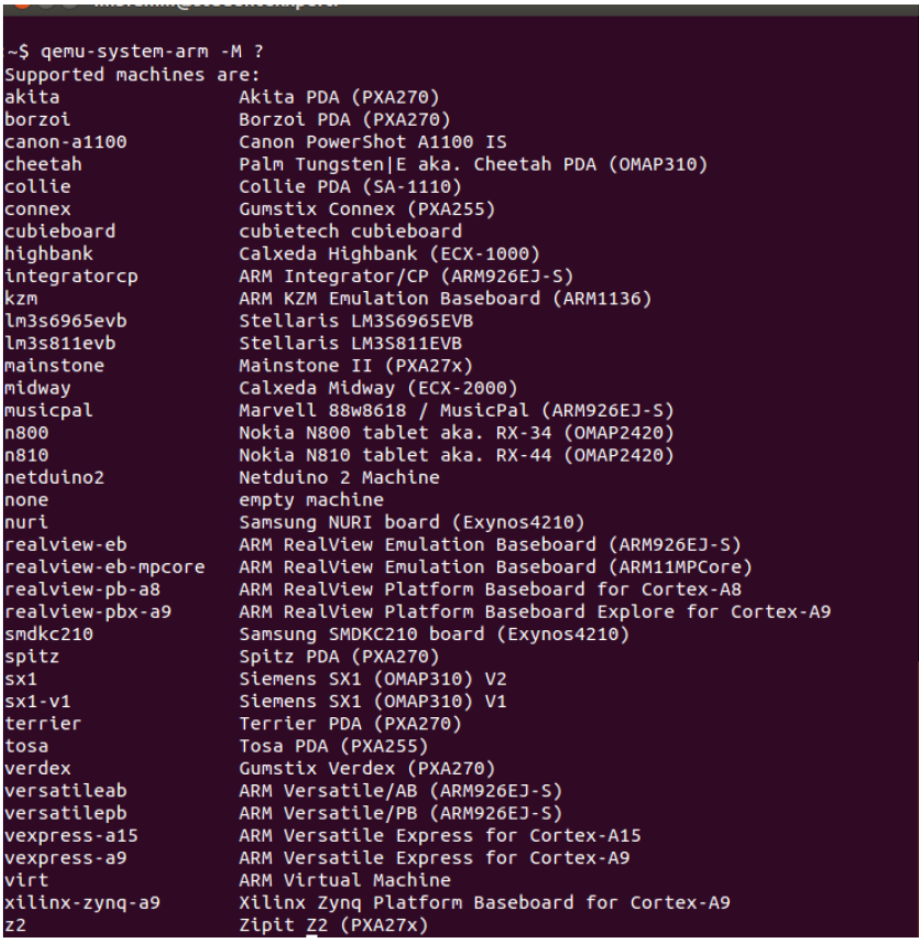
Figure 2: Supported Processors Samples

Since we target to work with development board, we need to make sure also that QEMU is supporting emulation of these boards. You can check that by typing

```
$qemu-system-arm -M ?
```

M denotes for machine, as development boards are defined as machines. You will get like figure 3.

It worth mentioning here that, to work with Raspberry board versatilepb board is used instead.



```
~$ qemu-system-arm -M ?
Supported machines are:
akita                Akita PDA (PXA270)
borzoi               Borzoi PDA (PXA270)
canon-a1100          Canon PowerShot A1100 IS
cheetah              Palm Tungsten|E aka. Cheetah PDA (OMAP310)
collie               Collie PDA (SA-1110)
connex               Gumstix Connex (PXA255)
cubieboard           cubietech cubieboard
highbank             Calxeda Highbank (ECX-1000)
integratorcp         ARM Integrator/CP (ARM926EJ-S)
kzm                  ARM KZM Emulation Baseboard (ARM1136)
lm3s6965evb          Stellaris LM3S6965EVB
lm3s811evb           Stellaris LM3S811EVB
mainstone            Mainstone II (PXA27x)
midway               Calxeda Midway (ECX-2000)
musicpal             Marvell 88w8618 / MusicPal (ARM926EJ-S)
n800                 Nokia N800 tablet aka. RX-34 (OMAP2420)
n810                 Nokia N810 tablet aka. RX-44 (OMAP2420)
netduino2            Netduino 2 Machine
none                 empty machine
nuri                 Samsung NURI board (Exynos4210)
realview-eb          ARM RealView Emulation Baseboard (ARM926EJ-S)
realview-eb-mpcore   ARM RealView Emulation Baseboard (ARM11MPCore)
realview-pb-a8       ARM RealView Platform Baseboard for Cortex-A8
realview-pbx-a9      ARM RealView Platform Baseboard Explore for Cortex-A9
smdkc210             Samsung SMDKC210 board (Exynos4210)
spitz                Spitz PDA (PXA270)
sx1                  Siemens SX1 (OMAP310) V2
sx1-v1               Siemens SX1 (OMAP310) V1
terrier              Terrier PDA (PXA270)
tosa                 Tosa PDA (PXA255)
verdex               Gumstix Verdex (PXA270)
versatileab          ARM Versatile/AB (ARM926EJ-S)
versatilepb          ARM Versatile/PB (ARM926EJ-S)
vexpress-a15         ARM Versatile Express for Cortex-A15
vexpress-a9          ARM Versatile Express for Cortex-A9
virt                 ARM Virtual Machine
xilinx-zynq-a9       Xilinx Zynq Platform Baseboard for Cortex-A9
z2                   Zipit Z2 (PXA27x)
```

Figure 3: Supported Machines Sample

If previous checks are met, everything shall be ready then for emulation. In this tutorial we will go through Raspberry Pi board emulation.

5 Get Hardware Specifications

From Raspberry website we can find that raspberry pi is using BCM2835/BCM2836 processor chip from Broadcom, follow further this chip you will find it is based on ARM1176 processor. Doing some online research you can find that this processor core is built on ARMv6 architecture.

To simplify it, all what we need to know from our Hardware: on which processor core and processor architecture it works.

6 Get Running Operating System

6.1 Image

For Raspberry pi, you can find ready builds "operating systems" that are designed to work with this board. You can find online these builds at the following link

http://www.raspberrypi.org/downloads/

Currently we will work with Raspbian, so download this image. This image is downloaded with size ~ 700 M. The emulation can't run on this space, so we have to append more disk space for this image. You can do that using qemu with the following command

```
$qemu-img resize 2015-02-16-raspbian-wheezy.img +2G
```

Image size will be around 2.7 G after this operation.

6.2 Kernel

Downloaded image "Raspbian" can't work standalone, you need to provide the kernel on which this image will run. Any Linux kernel should be suitable for running. So download any Linux kernel.

Operating System that you need to run should be also supporting working on ARM processors. To know that open the directory of your Linux version (directory of the Linux that you will run on the board) >> arch >> arm. You will find there folders with each supported machine. Since we target to emulate Raspberry, the Versatile machine should be supported. So in arm directory open folder "mach-versatile". Open "Kconfig" to know which models of the machine are supported.

Now step back up to arm directory then open folder "mm" then open file "Kconfig". You will find there all supported processors architecture by this Linux version. Make sure that your architecture is covered by the machine we will work with.

For example, In latest version downloaded from Linux kernel, ARMv6 wasn't covered by versatilepb machine. But our emulation needs it. So I added machine name "ARCH_VERSATILE_PB" in Kconfig file as shown in the following figure.

```
# ARMv6
config CPU_V6
	bool "Support ARM V6 processor" if ARCH_INTEGRATOR || MACH_REALVIEW_EB || MACH_REALVIEW_PBX || MACH_BCM2708 || ARCH_VERSATILE_PB || ARCH_VERSATILE_AB
	select CPU_32v6
	select CPU_ABRT_EV6
	select CPU_CACHE_V6
	select CPU_CACHE_VIPT
	select CPU_COPY_V6 if MMU
	select CPU_CP15_MMU
	select CPU_HAS_ASID if MMU
	select CPU_PABRT_V6
	select CPU_TLB_V6 if MMU

# ARMv6k
config CPU_V6K
```

Figure 4: Needed Kernel Configuration

If your kernel is supporting the versatile machine, you can continue now with compilation steps. In the following steps we will show how to prepare a kernel for raspberry emulation.

As any kernel, it should be compiled for the target platform. Compile means to convert system file (normally in C) to the suitable instruction set. That is why we have to make sure that the kernel is supporting target architecture. We need then a tool that compiles the system files to the target machine language. This tool is the cross compiler.

For Linux, you can work with a tool called codesourcery or linearo. In this tutorial we will work with codesourcery. You can download it from the link

https://sourcery.mentor.com/sgpp/lite/arm/portal/kbentry62

a file called "arm-2010.09-50-arm-none-linux-gnueabi" shall be downloaded, install it

```
$chmod +x arm-2010.09-50-arm-none-linux-gnueabi.bin
$./arm-2010.09-50-arm-none-linux-gnueabi.bin
```

Usually, you will get an error message tells that dash shell is not supported.

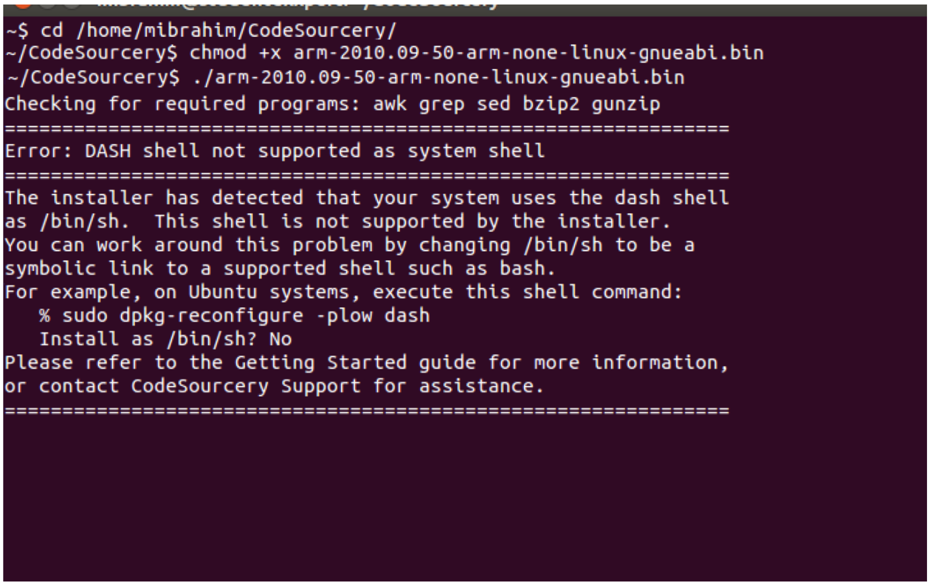


Figure 5: CodeSourcery Error

You can resolve this error by typing

```
$sudo dpkg-reconfigure -plow dash
```

then select No.

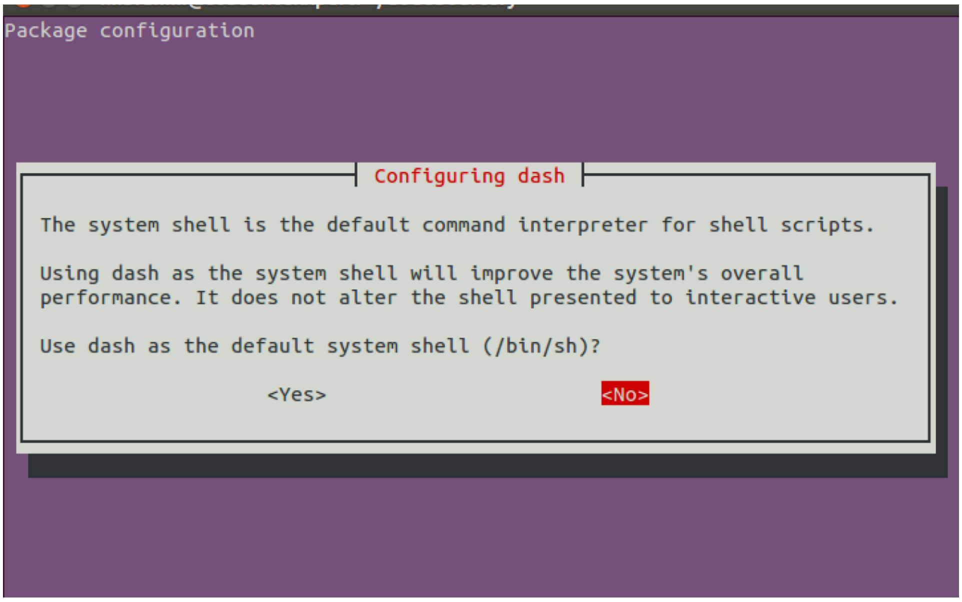


Figure 6: Change System Shell

Now, invoke the installation again

```
$./arm-2010.09-50-arm-none-linux-gnueabi.bin
```

Installation wizard shall start, follow steps to end of wizard.

When installation is completed, return to bash shell again

```
$sudo dpkg-reconfigure -plow dash
```

This time select Yes.

After you finish installation, you need to add executables of this codesourcery to bin directory in order to call it anywhere. So define its bin to PATH environment variables

```
$echo "export PATH=path/to/Sourcery_CodeBench_Lite_for_ARM_EABI/bin:\${PATH}"
    >> ~/.bashrc
```

To make sure that the cross compiler tool is ready, lets do small check.

Create a Hello World C code sample

```
int main (void) {
printf ("\n Hello World\n");
return 0;
}
```

Then compile this code by

```
$arm-none-linux-gnueabi-gcc -o hello hello.c
```

Then execute it by

```
$qemu-arm -L ~/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc hello
```

If you see Hello World message, then the cross compiler is successfully installed.

So far cross compile tool shall be successfully setup and ready. Remaining to configure and compile the kernel.

You can download Linux kernels from the following link

https://www.kernel.org/pub/linux/kernel/

unpack your downloaded package

```
$tar xvf linux-3.18.10.tar.bz2
$cd linux-3.18.10
$make ARCH=arm versatile_defconfig
$make ARCH=arm menuconfig
```

Configuration menu shall start now go to System setup to make sure that your kernel is supporting versatile machine

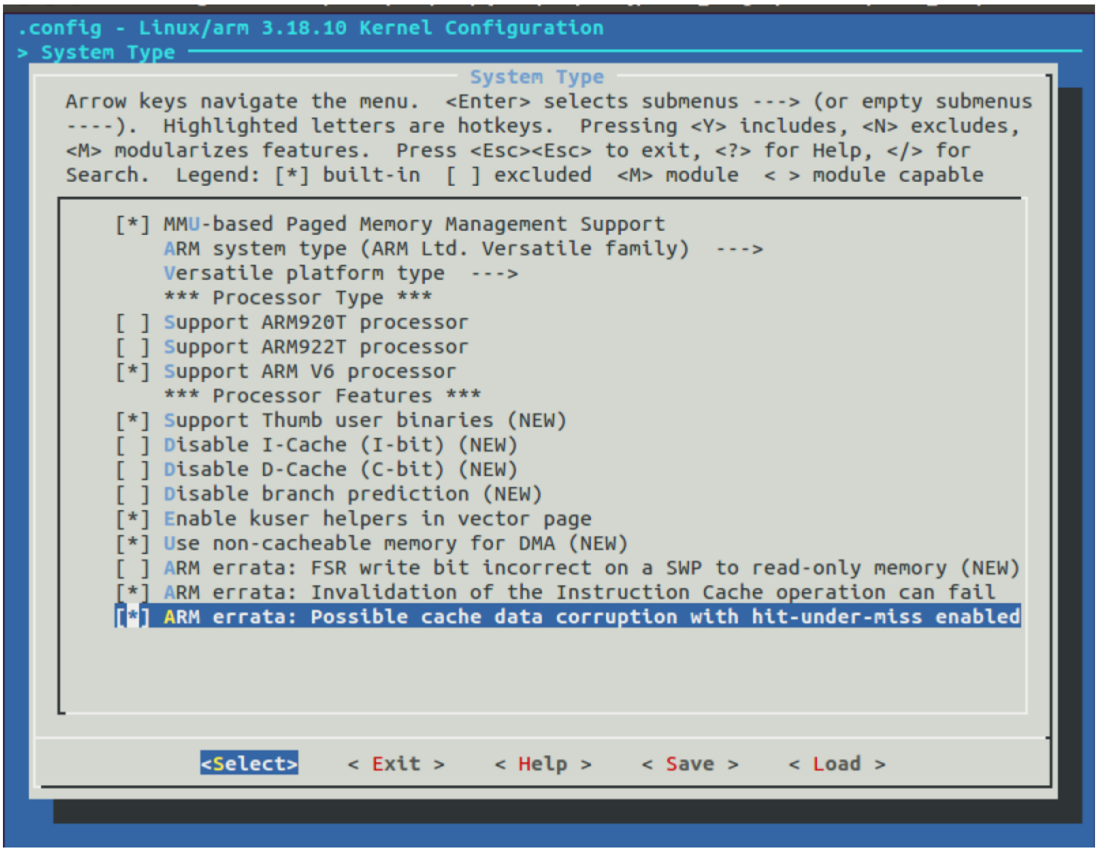


Figure 7: Kernel Menuconfig

Also in “General Setup” define the name of the cross compiler in “Cross-compiler tool prefix” option as “Arm-none-linux-gnueabi-” and don’t forget the last dash -. this is the name of codesourcery that we just installed. You can figure out this name from the bin directory of the installation directory.

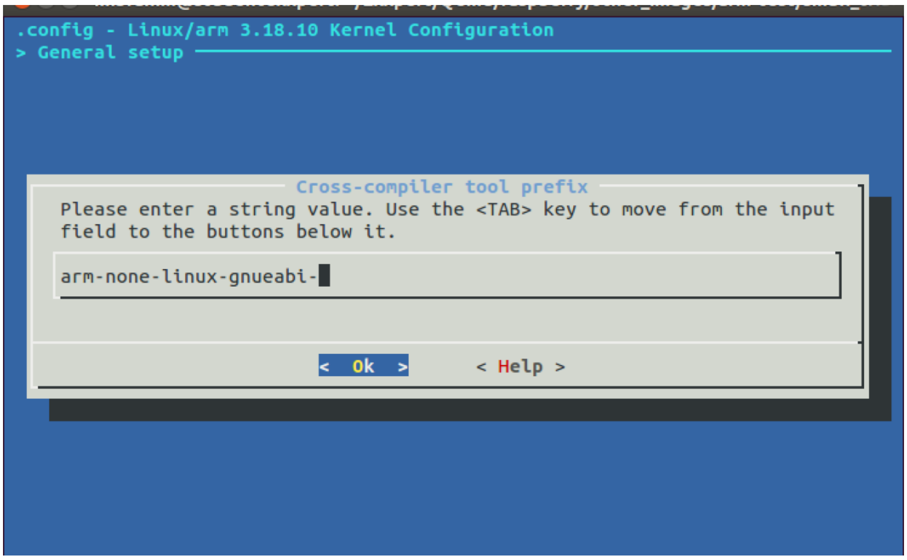


Figure 8: Cross Compiler Tool Prefix

Continue with other features setting as discussed in the following table

Credit for these configurations goes to http://xecdesign.com/compiling-a-kernel/

Section	Feature	Sub-Feature	Subsub-feature	Action
General Setup	Cross-compiler tool prefix			set its value to Arm-none-linux-gnueabi-
System Type	Support ARM V6 processor			Enable
	ARM errata: Invalidation of the Instruction Cache operation can fail			Enable
	ARM errata: Possible cache data corruption with hit-under-miss enabled			Enable
Floating point emulation	VFP-format floating point maths			Enable
Kernel Features	Use ARM EABI to compile the kernel			Enable
	Allow old ABI binaries to run with this kernel			Enable
Bus Support	PCI Support			Enable
Device Drivers	SCSI Device Support	SCSI Device Support		Enable
		SCSI Disk Support		Enable
		SCSI CDROM support		Enable
		SCSI low-lever drivers	SYM53C8XX Version 2 SCSI support	Enable
	Generic Driver Options	Maintain a devtmpfs filesystem to mount at /dev		Enable
		Automount devtmpfs at /dev, after the kernel mounted the root		Enable
	Input device support	Event interface		Enable
File systems	Ext3 journalling file system support			Enable
	The Extended 4 (ext4) filesystem			Enable
	Pseudo filesystems	Virtual memory file system support (former shm fs)		Enable

Table 1: Kernel Configuration

After this configuration step, your kernel is ready for compilation.

```
$make ARCH=arm
$make ARCH=arm INSTALL_MOD_PATH=../modules modules_install
```

To here you have compiled successfully a kernel for emulation usage. You can find an image created for this kernel in the directory arch/arm/boot, there you can find zImage of the kernel. We will use this image to run the emulation, so take a copy.

7 Start the Emulation

All prerequisites are now prepared. Emulation is ready to start. First, we start with some fixes in the downloaded image to work with the emulation. So start the emulation as a shell display by executing

```
$qemu-system-arm -kernel kernel-qemu -cpu arm1176 -m 256 -M versatilepb -append
    "root=/dev/sda2 panic=1 init=/bin/sh rw" -hda 2015-02-16-wheezy-raspbian.
    img
```

When shell prompt becomes ready type

```
$nano /etc/ld.so.preload
```

A file will open. It contains exactly one line. Add # to this line to be

```
#/usr/lib/arm-linux-gnueabihf/libcofi_rpi.so
```

Now press Ctrl+O <ENTER> to write the file and then Ctrl+X to exit the editor.

Another file that is needed to be created, type

```
$nano /etc/udev/rules.d/90-qemu.rules
```

Add the following in that file

```
KERNEL=="sda", SYMLINK+="mmcblk0"
KERNEL=="sda?", SYMLINK+="mmcblk0p%n",
```

Don't forget the last comma. Then press Ctrl+O <ENTER> to write the file and then Ctrl+X to exit the editor.

Up to here all fixes needed for the image to run on qemu has been applied. We can now start running the system on Qemu. Close current Qemu window and write back in Terminal window the following

```
$qemu-system-arm -kernel kernel-qemu -cpu arm1176 -m 256 -M versatilepb -append
    "root=/dev/sda2 panic=1" -hda 2015-02-16-wheezy-raspbian.img
```

It will boot to raspi-config as shown in figure 9. Click Finish

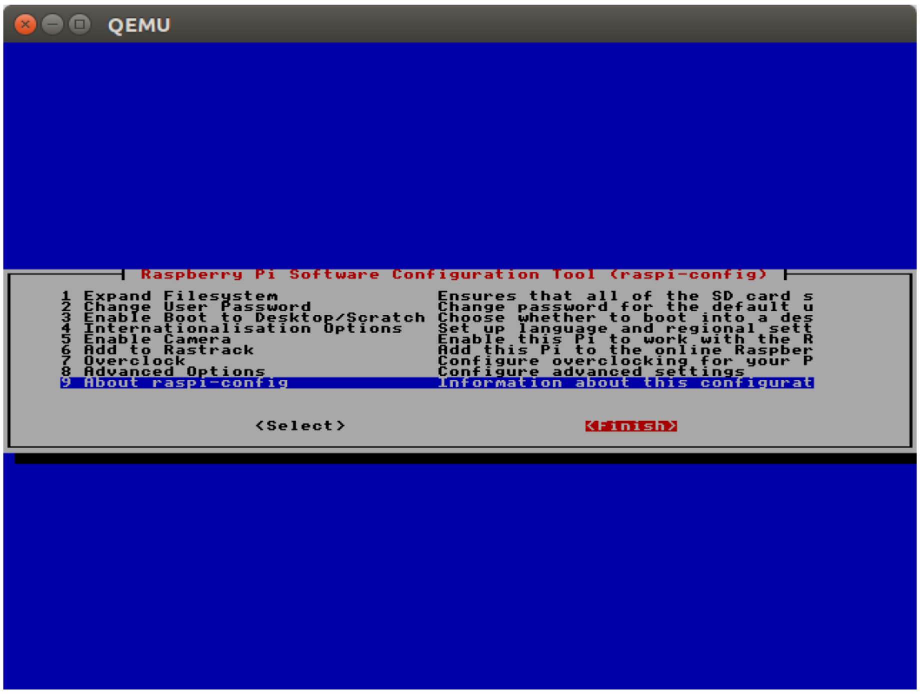


Figure 9: Raspi-config in QEMU

Create a link with the following command

```
$sudo ln -snf mmcblk0p2 /dev/root
```

then start raspi-config with

```
$sudo raspi-config
```

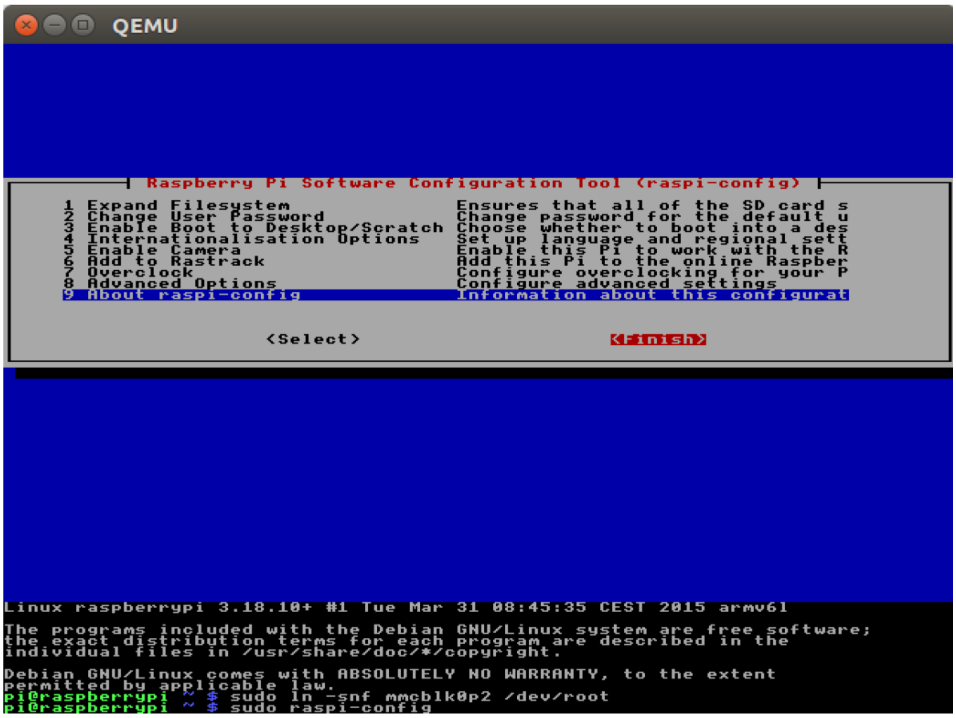


Figure 10: Create SymLink to Booting Device

Select first option

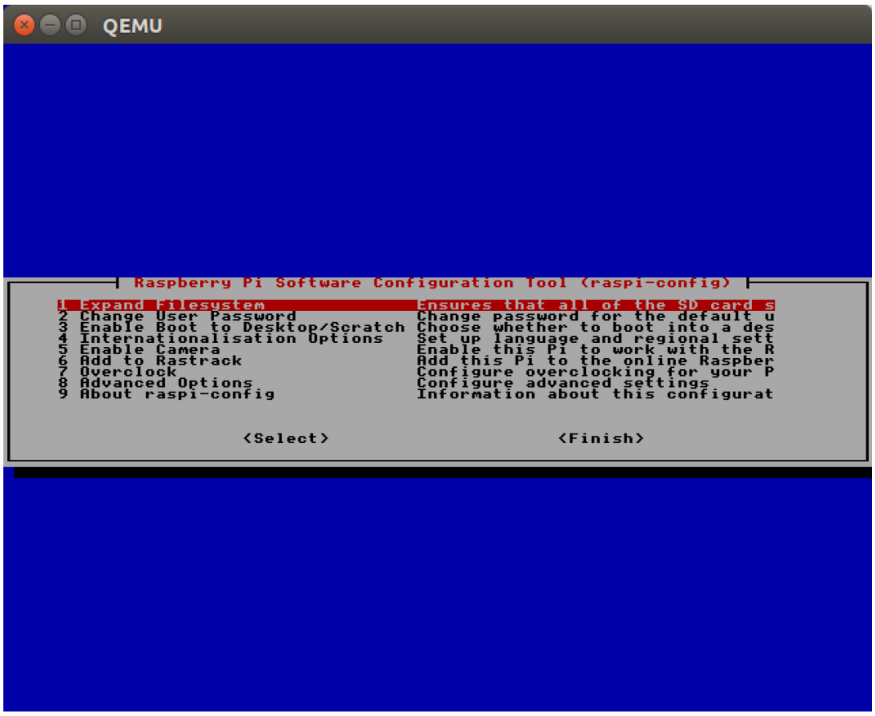


Figure 11: Expand File System for Raspberry

Then you will have the following screen, select ok

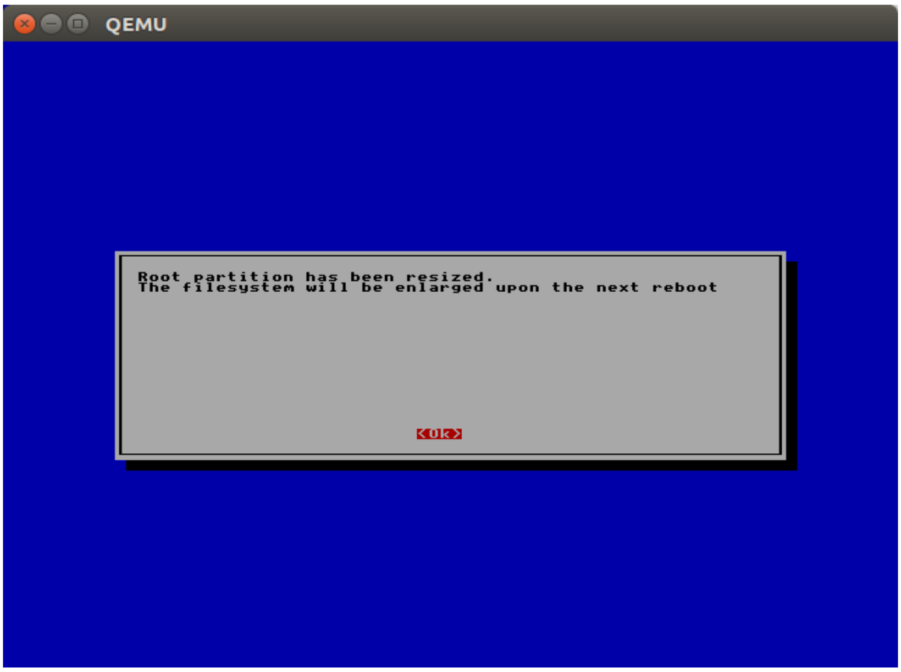


Figure 12: Expand File System Confirmation

Select Finish

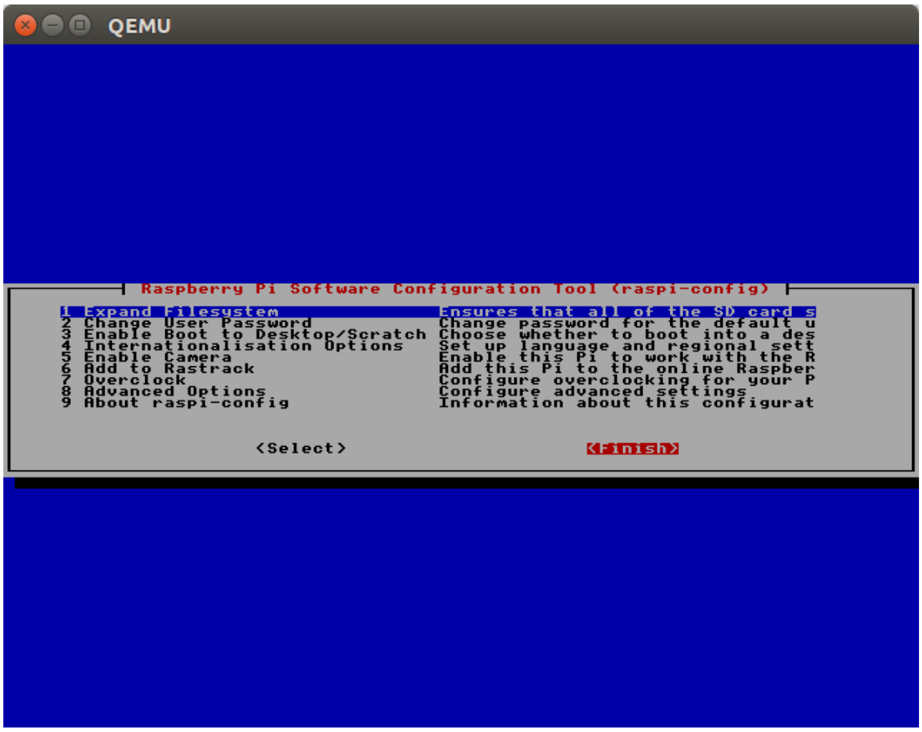


Figure 13: Finish Expand File System

Now select Yes to reboot

Figure 14: Reboot Configuration

Login to Raspbian with the following credentials
username: pi, password: raspberry
These credentials are given in Raspberry Pi website.

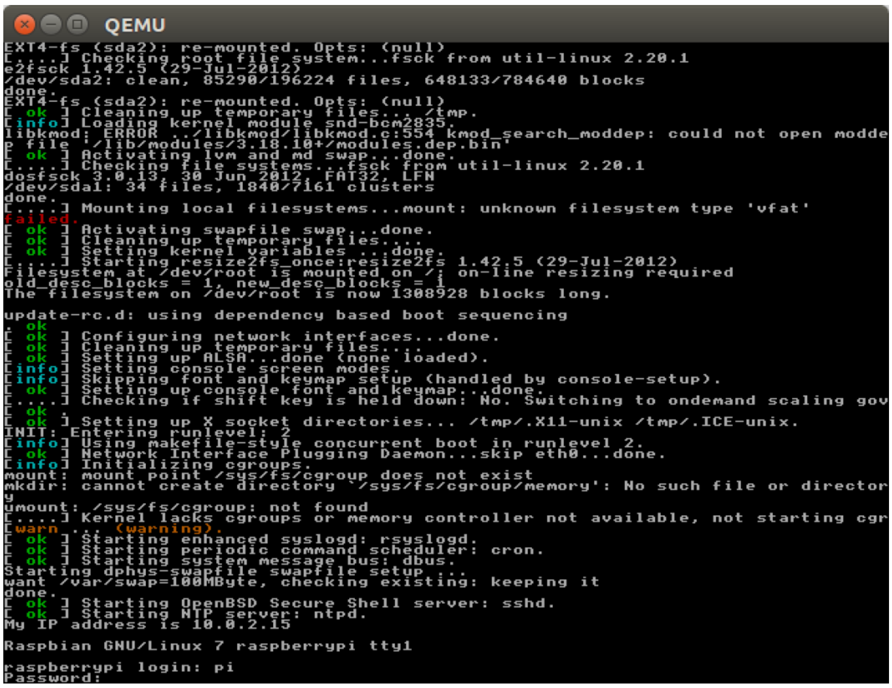


Figure 15: Raspbian Login

We need to do some fixes for the emulation display, write the following command

```
$sudo nano /etc/X11/xorg.config
```

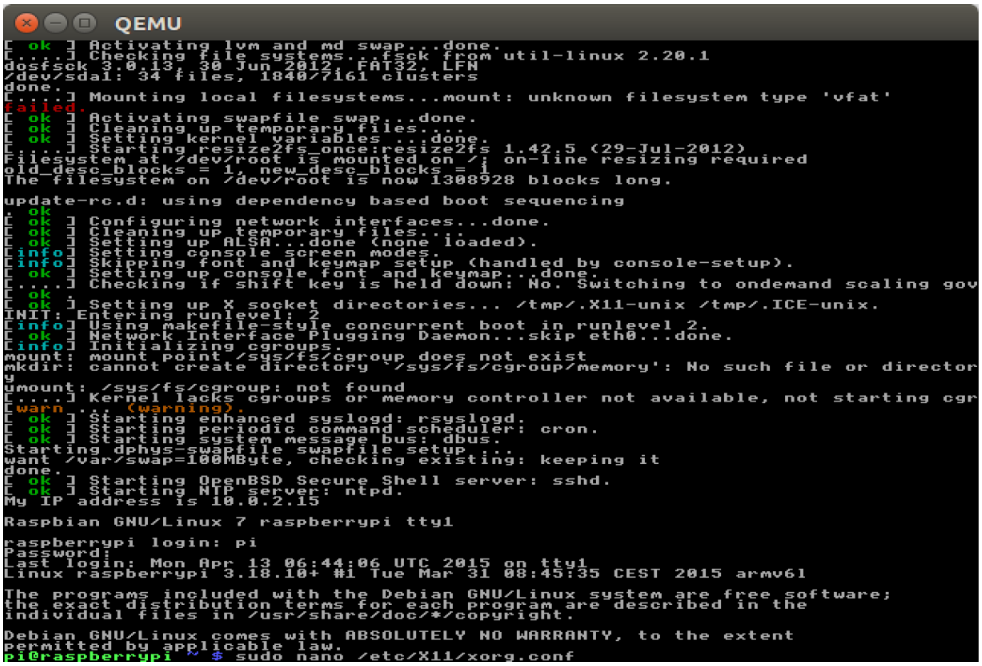


Figure 16: Fix Display Options

Then add the following lines

```
Section "Screen"
Identifier "Default Screen"
SubSection "Display"
Depth 16 Modes "800x600" "640x480"
EndSubSection
EndSection
```

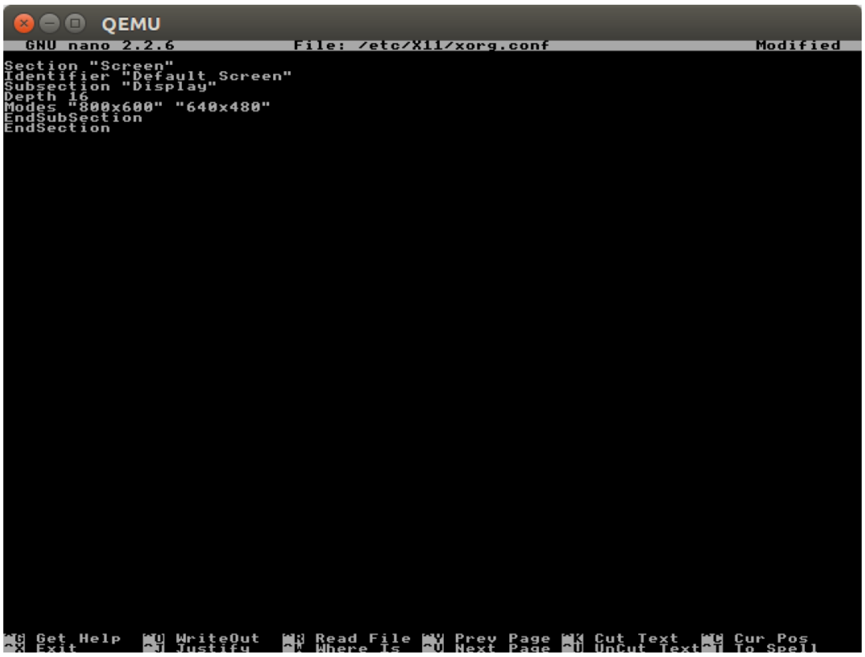


Figure 17: Display Settings

Then CTRL+O then ENTER then CTRL+X to save and exit.

Finally you can start gui of raspberry, type startx

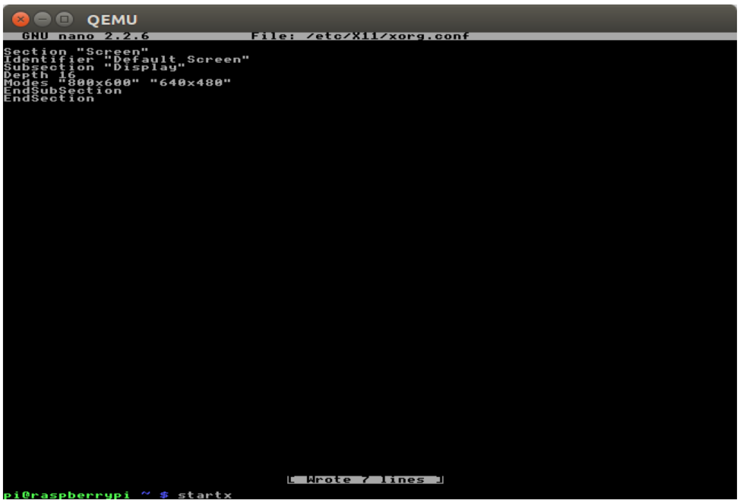


Figure 18: Starting Raspbian GUI

Enter

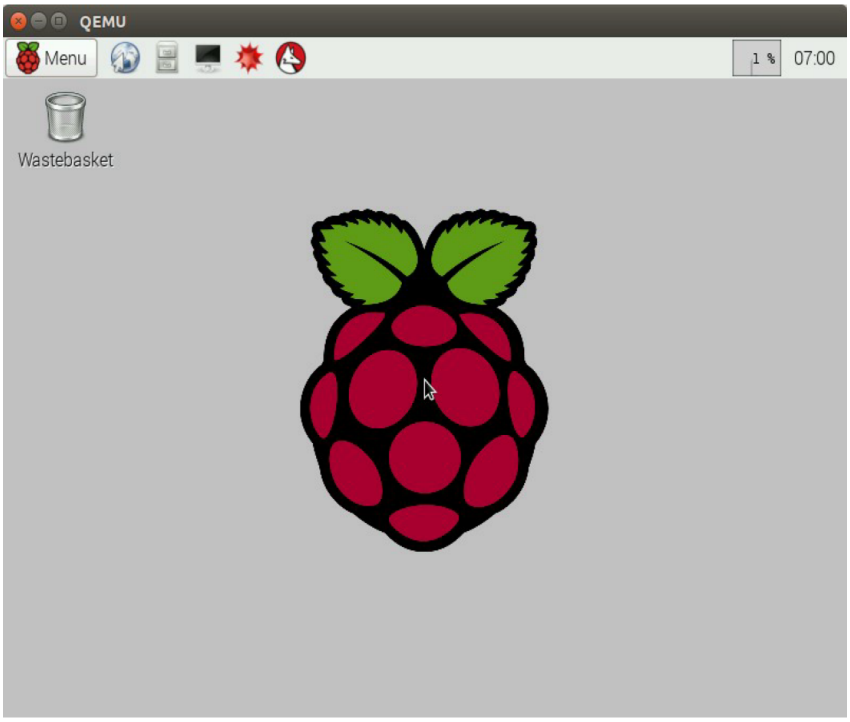


Figure 19: Raspbian Desktop

Bibliography

[1] http://xecdesign.com/qemu-emulating-raspberry-pi-the-easy-way/

[2] http://xecdesign.com/compiling-a-kernel/

[3] http://www.elinux.org/Virtual_Development_Board

[4] https://balau82.wordpress.com/arm-emulation/

[5] https://sourcery.mentor.com/sgpp/lite/arm/portal/kbentry62

[6] https://www.cs.umd.edu/~meesh/cmsc411/website/proj01/arm/

[7] https://darrenjw2.wordpress.com/2015/02/07/getting-started-with-snappy-ubuntu-core-on-the- raspberry-pi-2/

[8] http://www.pezzino.ch/sourcery-codebench-lite/

[9] https://gist.github.com/bdsatish/7476239

[10] http://www.slideshare.net/sherif_mosa/building-embedded-linux-full-tutorial-for-arm