

1.) Prepare the x86\_64 Debian Host

`mkdir /home/youruser/assets` this will be the target for the final image

Install all required packages for QEMU

`sudo apt install qemu-efi-aarch64 qemu-system-arm virt-manager`

Download the arm64 mini.iso from Debian

<https://d-i.debian.org/daily-images/arm64/daily/netboot/>

2.) Setup Virtual Machine in QEMU

open Virtual Machine Manager

select "Local install media (ISO image or CDROM)"

in "Architecture options" select Architecture: **aarch64** and Machine Type: **virt**

next select the just downloaded **mini.iso**

next choose the operating system **Debian 10**

next set Memory to **1024** and CPUs to **4**

next create a disk image and set size to **4 GiB**

finally click "Finish" and click "Yes" to make Virtual Network active

3.) Install Debian for arm64 in your Virtual Machine

click into the black area of the VMs Window to capture Mouse and Keyboard

hit Enter to start text based Debian Installer

create **root** password and **youruser** with password as they will be on the final image

partition manually the disk image as follows

Partition 1: Size **100M**, Name **efi**, Use as **EFI System Partition**, Bootable flag on

Partition 2: Size **100M**, Name **boot**, Use as **Ext 2 file system**, Mount point **/boot**

Bootable flag off

Partition 3: Size **max**, Use as **Ext 4 journaling file system**, Mount point **/**

Bootable flag off

confirm that you don't want to create Swap Space by clicking **<NO>**

in "Software selection" select only **SSH server** and **standard system utilities**

and finish the installation, once finished reboot into the newly installed system

4.) DTB file handling

`mkdir /boot/dtbs`

`nano /etc/kernel/postinst.d/copy-dtbs`

`#!/bin/sh`

`set -e`

`version="$1"`

`echo Copying current dtb files to /boot/dtbs....`

`cp -a /usr/lib/linux-image-`${version}`/. /boot/dtbs/`

`chmod +x /etc/kernel/postinst.d/copy-dtbs`

`/etc/kernel/postinst.d/copy-dtbs `uname -r``

5.) Bootloader configuration

```
mkdir /boot/extlinux
```

```
nano /boot/extlinux/extlinux.conf
```

```
TIMEOUT 2
```

```
PROMPT 1
```

```
DEFAULT debian
```

```
LABEL debian
```

```
MENU LABEL Debian
```

```
KERNEL /vmlinuz
```

```
INITRD /initrd.img
```

```
DEVICETREEDIR /dtbs
```

```
APPEND console=tty1 console=ttyS2,1500000 root=LABEL=root rw rootwait
```

```
apt purge grub-efi-arm64
```

```
apt autoremove
```

```
apt autoclean
```

```
shutdown -h now
```

6.) Creating tar archives of our VM

```
sudo modprobe nbd max_part=8
```

```
sudo qemu-nbd --connect=/dev/nbd0 /var/lib/libvirt/images/debian10-aarch64.qcow2
```

```
sudo mount /dev/nbd0p2 /mnt
```

```
cd /mnt
```

```
sudo tar cfvzp /home/youruser/assets/debian-aarch64-bootfs.tar.gz .
```

```
cd
```

```
umount /mnt
```

```
sudo mount /dev/nbd0p3 /mnt
```

```
cd /mnt
```

```
sudo tar cfvzp /home/youruser/assets/debian-aarch64-rootfs.tar.gz .
```

```
cd
```

```
umount /mnt
```

```
sudo qemu-nbd -d /dev/nbd0
```

7.) Install Cross Compiler for building U-Boot on our x86\_64 Debian Host

```
sudo apt install device-tree-compiler build-essential libssl-dev python3-dev bison
```

```
sudo apt install flex libssl-dev swig gcc-aarch64-linux-gnu gcc-arm-none-eabi
```

```
(sudo apt install gcc make bc git)
```

8.) Build U-Boot on our x86\_64 Debian Host

```
git clone https://github.com/ARM-software/arm-trusted-firmware
```

```
cd arm-trusted-firmware
```

```
git tag
```

remember last stable (v2.3)

```
git checkout v2.3
```

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=rk3328 bl31
```

```
cd ..
```

```

git clone git://git.denx.de/u-boot.git
cd u-boot
git tag
git checkout v2020.07
ln -s /home/youruser/arm-trusted-firmware/build/rk3328/release/bl31/bl31.elf bl31.elf
make CROSS_COMPILE=aarch64-linux-gnu- BL31=bl31.elf rock64-rk3328_defconfig
make -j4 CROSS_COMPILE=aarch64-linux-gnu- BL31=bl31.elf all u-boot.itb

cp /home/youruser/u-boot/idbloader.img /home/youruser/assets/
cp /home/youruser/u-boot/u-boot.itb /home/youruser/assets/

```

## 9.) Assembling the final image for our Pine64 Rock64 SBC

```

sudo apt install kpartx

cd /home/youruser/assets
dd if=/dev/zero of=debian-rock64.img bs=1M count=4096

nano sfdisk.template

label: mbr
unit: sectors
first-lba: 64

start= 2048, size= 16384
start= 18432, size= 614400, bootable
start= 632832

sudo /sbin/sfdisk debian-rock64.img < sfdisk.template
sudo kpartx -v -a debian-rock64.img

sudo mkfs.ext2 -m0 -L boot /dev/mapper/loop0p2
sudo mount /dev/mapper/loop0p2 /mnt
cd /mnt
sudo tar xzvpf /home/youruser/assets/debian-aarch64-bootfs.tar.gz .
sync
cd
sudo umount /mnt

sudo mkfs.ext4 -L root /dev/mapper/loop0p3
sudo mount /dev/mapper/loop0p3 /mnt
cd /mnt
sudo tar xzvpf /home/youruser/assets/debian-aarch64-rootfs.tar.gz .
sync
cd
sudo umount /mnt

cd /home/youruser/assets/
dd if=idbloader.img of=debian-rock64.img seek=64 conv=notrunc
dd if=u-boot.itb of=debian-rock64.img seek=16384 conv=notrunc

```

## 10.) Flashing the newly build image onto eMMC-Module for our Pine64 Rock64 SBC

```

lsblk
(sudo umount /dev/sdX1)

cd /home/youruser/assets/
sudo dd if=debian-rock64.img of=/dev/sdX bs=1M
cd

```

- 11.) Use GParted and create a SWAP partition of 1GB at the end of the eMMC-Module and then extend the /root partition to fill up the empty space between. (leave 1MB as free space at the end of eMMC)
- 12.) Installing the eMMC-Module onto your Pine64 Rock64 SBC, connecting HDMI, Mouse and Keyboard and power it up.

`nano /etc/network/interfaces`

change interface to eth0

# This file describes the network interfaces available on your system  
# and how to activate them. For more information, see interfaces(5).

`source /etc/network/interfaces.d/*`

# The loopback network interface  
`auto lo`  
`iface lo inet loopback`

# The primary network interface  
`allow-hotplug eth0`  
`iface eth0 inet dhcp`

`nano /etc/fstab`

add SWAP and replace Labels with device names

# /etc/fstab: static file system information.  
#  
# Use 'blkid' to print the universally unique identifier for a  
# device; this may be used with UUID= as a more robust way to name devices  
# that works even if disks are added and removed. See fstab(5).  
#  
# systemd generates mount units based on this file, see systemd.mount(5).  
# Please run 'systemctl daemon-reload' after making changes here.  
#  
# <file system> <mount point> <type> <options> <dump> <pass>  
/dev/mmcblk1p2 /boot ext2 defaults 0 2  
/dev/mmcblk1p3 / ext4 errors=remount-ro 0 1  
/dev/mmcblk1p4 swap swap defaults 0 0  
/dev/sr0 /media/cdrom0 udf,iso9660 user,noauto 0 0

`reboot`

`ip a`

check that network is working

`apt update`

perform system update

`apt upgrade`

`apt dist-upgrade`

`apt autoremove`

`apt autoclean`

Done, enjoy your setup.

