RPi U-Boot

From eLinux.org
Jump to: navigation, search

Back to the Hub.

Software & Distributions:

Software - *an overview.*

Distributions - *operating systems and development environments for the Raspberry Pi.*

Kernel Compilation - *advice on compiling a kernel.*

Performance - *measures of the Raspberry Pi's performance.*

Programming - *programming languages that might be used on the Raspberry Pi.*

Contents

[hide]

Overview

Das U-Boot, often abbreviated to just U-Boot, is a bootloader commonly used on embedded systems. U-Boot can be used on the RPi to add flexibility by allowing other boot configurations to be used on top of the single specified file on the SD card. If you wish to run an upstream kernel, booting it via U-Boot is recommended.

This page explains how to get U-Boot working on the RPi, and explains how to boot images off both the SD card and over TFTP from the network.

Get the source

Various forks of U-Boot contain (different levels of) support for the Raspberry Pi. You will to download the source from one of the locations below; pick one, and ignore the others.

At this point in time, mainline U-Boot is recommended for all purposes; all features should be present in mainline U-Boot now. The only exception is if you're actively collaborating with another developer on a specific feature.

Mainline

The primary repository for U-Boot is git://git.denx.de/u-boot.git master branch. At this point in time, you should expect to use this repository for all purposes. This version supports the UART (serial port), SD card, HDMI display, and USB high-speed devices. USB low-/full-speed (USB 1.0) devices do not currently work. USB Ethernet devices are typically high-speed and hence work, and USB keyboards are typically low-/full-speed devices and hence do not currently work.

To download this repository into a new local repository (this is what you want if you don't know better):

```
git clone git://git.denx.de/u-boot.git
```

To add this repository to an existing local repository:

```
git remote add u-boot git://git.denx.de/u-boot.git
git fetch u-boot
git checkout -b your_branch_name u-boot/master
```

Stephen Warren's work-in-progress branch

Stephen Warren works on RPi support in mainline U-Boot. Some patches will appear in his work-in-progress repository before they get upstream. His work-in-progress repository is git://github.com/swarren/u-boot.git branch rpi_dev. Note that this branch is often rebased, and almost certainly contains work other than mainline-quality RPi-related patches.

To download this repository into a new local repository (this is what you want if you don't know better):

```
git clone git://github.com/swarren/u-boot.git
git checkout -b rpi_dev origin/rpi_dev
```

To add this repository to an existing local repository:

```
git remote add github_swarren_u-boot git://github.com/swarren/u-boot.git
git fetch github_swarren_u-boot
git checkout -b rpi_dev github_swarren_u-boot/rpi_dev
```

Oleksandr Tymoshenko's branch

Oleksandr's branch contains working USB support. Pre-built tarballs can be found here. These releases are meant to boot FreeBSD; if you're booting Linux, copy only the u-boot binary to your boot partition and follow the instructions later on this page.

To build it yourself, get the source from git://github.com/gonzoua/u-boot-pi.git branch rpi.

To download this repository into a new local repository (this is what you want if you don't know better):

```
git clone -b rpi git://github.com/gonzoua/u-boot-pi.git
```

To add this repository to an existing local repository:

```
git remote add github_gonzoua_u-boot-pi git://github.com/gonzoua/u-boot-pi.git
git fetch github_gonzoua_u-boot-pi
git checkout -b your_branch_name github_gonzoua_u-boot-pi/rpi
```

You can download a source tarball from this link.

Marco Schuster's branch

This branch is based off mainline HEAD and adds FDT/ATAG cmdline passthrough; this makes loading and compositing the FDT yourself in u-boot unneccessary.

To build it, go to https://github.com/msmuenchen/u-boot. The debian_rpi branch contains a replacement debian/ packaging directory.

To build a Debian package (recommended to do on a RPi or a qemu-static buildroot, cross-compiling is untested and unsupported), run

```
debian/rules binary
```

inside the directory and install the debian package built one level up. This will automatically modify config.txt to boot with u-boot and always boot from the kernel with the mpst current version number, provided it has been installed using one of the standard kernel images (linux-image-rpi-rpfv/linux-image-rpi2-rpfv).

A working uboot.env is supplied with the package; take care that it will be overwritten on reinstalls. fw_printenv/fw_setenv are installed together with a fw_env.config.

Get an ARM compiler

If you are building U-Boot on a Raspberry Pi, or other ARM platform, you can skip this step; the native compiler already generates ARM instructions.

If you are building U-Boot on a different system, e.g. an x86 PC, you will need an ARM cross-compiler. Pick one of the options below, and ignore the others.

The toolchain you pick needs to have its associated libgcc compiled in a manner that is compatible with the RPi's ARM CPU. If this is not the case, see the section below that describes the Linaro compiler.

Ubuntu packages

(Stephen Warren recommends this option)

Ubuntu includes an ARM cross compiler in its standard package step. You can install it as follows:

```
apt-get install gcc-arm-linux-gnueabi
```

To build u-boot in 64-bit mode for RPi3, install the 64-bit compiler:

```
apt-get install gcc-aarch64-linux-gnu
```

Each time you start a new shell to compile U-Boot, run the following first:

```
export CROSS_COMPILE=arm-linux-gnueabi-
```

or, for the 64-bit compiler:

```
export CROSS_COMPILE=aarch64-linux-gnu-
```

Other distributions likely also provide ARM cross-compilers. Consult distro-specific documentation for details.

Raspberry Pi Foundation Tools

The Raspberry Pi Foundation provides a compiler in their tools repository. You can obtain it by:

```
cd $HOME
git clone git://github.com/raspberrypi/tools rpi-tools
```

Each time you start a new shell to compile U-Boot, run one of the following first, depending on which particular toolchain you want to use:

```
export CROSS_COMPILE=$HOME/rpi-tools/arm-bcm2708/arm-bcm2708hardfp-linux-gnueabi/bin/arm-bcm2708hardfp-linux-gnueabi-

export CROSS_COMPILE=$HOME/rpi-tools/arm-bcm2708/arm-bcm2708-linux-gnueabi/bin/arm-bcm2708-linux-gnueabi-

export CROSS_COMPILE=$HOME/rpi-tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/arm-linux-gnueabihf/bin/
```

Linaro Compilers

The Linaro compilers (at least those current as of 20140210) contain a non-multilib libgcc that is compiled for a newer CPU architecture than the RPI's CPU. Hence, by default, the Linaro compiler will generate a U-Boot image that doesn't work. To solve this, run the following before compiling U-Boot:

```
export USE_PRIVATE_LIBGCC=yes
```

You can also create or edit the file `arch/arm/cpu/arm1176/bcm2835/config.mk`, with the following content:

```
USE_PRIVATE_LIBGCC = yes
```

Make sure you perform a clean build after making setting `USE_PRIVATE_LIBGCC`.

QEMU-based chroot

This is a bit more complex, but especially useful if you have problems setting up your toolchain. You will need a Debian host system with the following packages:

```
coreutils kpartx e2fsprogs dosfstools mount tar util-linux debootstrap qemu-user-static
```

Then, build yourself a chroot:

```
debootstrap --no-check-gpg --foreign --arch armhf stretch /opt/buildroot/ http://mirrordirector.raspbian.org/raspbian
cp /usr/bin/qemu-arm-static /opt/buildroot/usr/bin
mount /dev /opt/buildroot/dev -o bind
mount /dev/pts /opt/buildroot/dev/pts -o bind
mount /sys /opt/buildroot/sys -o bind
mount /proc /opt/buildroot/proc -o bind
chroot /opt/buildroot/ /debootstrap/debootstrap --second-stage
(wait, this will take time)
(check using mount if all the bindmounts are still present. re-bind them if they're not, in case debootstrap has unmounted them)
chroot /opt/buildroot vi /etc/apt/sources.list
(remove the default, set deb http://mirrordirector.raspbian.org/raspbian main contrib non-free and deb-src http://mirrordirector.raspbian.org/raspbian mai
chroot /opt/buildroot/ apt-get update
chroot /opt/buildroot/ apt-get install build-essential
chroot /opt/buildroot/ apt-get build-dep u-boot
```

When you want to enter the chroot, run

```
chroot /opt/buildroot /bin/bash (this enters a shell where you can do compilations etc)
```

Note that prior to entering you have to bind-mount and after leaving unbind the mounts. Otherwise YOU WILL RUN INTO TROUBLE.

Compile the source

(Remember to set up CROSS_COMPILE first, as described above.)

Then, in the U-Boot source directory, run one of the following to configure the source tree to build for the Raspberry Pi:

For current version of U-Boot as of 17.8.2015

```
make rpi_defconfig
```

or

```
make rpi_2_defconfig
```

For RPi3 (remember to set CROSS_COMPILE to the 64-bit compiler)

```
make rpi_3_defconfig
```

Older versions of U-Boot:

```
make rpi_b_defconfig
```

even older versions:

```
make rpi_b_config
```

Then, perform the actual build:

```
make -j8 -s
```

You may wish to change the "8" in "-j8" to match the number of CPUs on your build system, or whatever make job limit you find works best.

The build process should take no more than a few minutes. This should generate `u-boot.bin`.

Copy U-Boot to your SD card

The RPi binary firmware can boot U-Boot just like it can boot a downstream kernel. As such, you can simply copy `u-boot.bin` to `kernel.img` on the SD card. You shouldn't need any options in config.txt for this to work; in fact, you may simply want to delete config.txt completely (but keep a backup!). Something like the following should work, as root:

Once, to back up your original config.txt:

```
mount /dev/sdb1 /mnt/tmp
mv /mnt/tmp/config.txt /mnt/tmp/config.txt.pre-uboot
umount /mnt/tmp
```

Every time you want to copy a new U-Boot binary to the SD card:

```
mount /dev/sdb1 /mnt/tmp
cp u-boot.bin /mnt/tmp/kernel.img
umount /mnt/tmp
```

For RPi3, note that there may be several kernel images on the SD card (e.g. Raspian images provide kernel.img, kernel7.img) and the boot firmware will pick the highest numbered one. So, for RPi3, the default kernel.img can be left, and u-boot.bin can be copied to a higher numbered kernel image so it takes precedence over the others:

```
mount /dev/sdb1 /mnt/tmp
cp u-boot.bin /mnt/tmp/kernel8.img
umount /mnt/tmp
```

Alternatively, you may wish not to rename the U-Boot binary. If so, the following should work, as root:

(**IMPORTANT**: the following is known not to work on RPi3 with 64-bit u-boot.bin)

Once, to back up your original config.txt:

```
mount /dev/sdb1 /mnt/tmp
mv /mnt/tmp/config.txt /mnt/tmp/config.txt.pre-uboot
umount /mnt/tmp
```

Every time you want to copy a new U-Boot binary to the SD card:

```
mount /dev/sdb1 /mnt/tmp
cp u-boot.bin /mnt/tmp/
echo 'kernel=u-boot.bin' > /mnt/tmp/config.txt
umount /mnt/tmp
```

In the commands above, you may need to replace "/dev/sdb1" with a different device filename, depending on where your SD card reader shows up. `cat /proc/partitions` should help you locate the correct device filename.

Test U-Boot

Boot up the RPi with your new U-Boot image. You should see U-Boot load on the screen (HDMI), and also on the serial port output (if you have it connected up).

Kernel Command-Line

In all of the example commands shown below to boot a kernel:

- You may omit `earlyprintk` if you're not doing kernel development/testing, and are not experiencing early boot problems.
- You may omit `console=ttyAMA0` if you don't want to use the serial console.
- You may omit `console=tty0` if you don't want to use a virtual terminal console (HDMI/USB keyboard).
- The `root=` option is correct for a Raspbian image. You may need to adjust this if your partition layout is different.
- The same applies for `rootfstype=` option. The default FS type is EXT4. You will need to change it if using a different filesystem.

Booting from an SD card

If your kernel uses Device Tree (it is true for a default upstream kernel image):

```
# swarren's branch already sets this automatically, so you can skip this
# Mainline U-Boot will set the following automatically soon
setenv fdtfile bcm2835-rpi-b.dtb

mmc dev 0
fatload mmc 0:1 ${kernel_addr_r} zImage
# IMPORTANT NOTE: On mainline u-boot, the correct variable to use here is ${fdt_addr} and NOT ${fdt_addr_r}
fatload mmc 0:1 ${fdt_addr_r} ${fdtfile}
setenv bootargs earlyprintk console=tty0 console=ttyAMA0 root=/dev/mmcblk0p2 rootfstype=ext4 rootwait noinitrd
# IMPORTANT NOTE: On mainline u-boot, the correct variable to use here is ${fdt_addr} and NOT ${fdt_addr_r}
bootz ${kernel_addr_r} - ${fdt_addr_r}
```

Note regarding ${fdt_addr_r} and ${fdt_addr}: In mainline U-Boot, the bootloader has been modified to not touch the FDT prepared by the RPi first stage bootloader. The address of this FDT is stored in the variable ${fdt_addr}. Thus, if you want to rely on the first stage bootloader to prepare the FDT, you can ignore the line **fatload mmc 0:1 ${fdt_addr_r} ${fdtfile}**, which loads the FDT from a file, simply load the kernel as usual and then call **bootz ${kernel_addr_r} - ${fdt_addr}**.

To boot a kernel that doesn't use Device Tree:

```
mmc dev 0
fatload mmc 0:1 ${kernel_addr_r} zImage
setenv bootargs earlyprintk console=ttyAMA0 console=tty1 root=/dev/mmcblk0p2 rootwait
bootz ${kernel_addr_r}
```

Network boot via TFTP

Configure DHCP server

If you don't want to use a static IP for the RPi, you'll need to make sure it's attached to a network with an active DHCP server.

Configure TFTP Server

You will need to have a TFTP server installed and configured.Place your zImage and DTB file(s) in the TFTP root directory, and ensure that the file permissions are set accordingly (everyone should have read access - use `chmod a+r zImage` if unsure).

Boot Using Dynamic IP

To boot a kernel that doesn't use Device Tree:

```
usb start
dhcp ${kernel_addr_r} zImage
setenv bootargs earlyprintk console=ttyAMA0 console=tty1 root=/dev/mmcblk0p2 rootwait
bootz ${kernel_addr_r}
```

If your kernel uses Device Tree:

```
# swarren's branch already sets this automatically, so you can skip this
# Mainline U-Boot will set the following automatically soon
setenv fdtfile bcm2835-rpi-b.dtb

usb start
dhcp ${kernel_addr_r} zImage
tftp ${fdt_addr_r} ${fdtfile}
setenv bootargs earlyprintk console=ttyAMA0 console=tty1 root=/dev/mmcblk0p2 rootwait
bootz ${kernel_addr_r} - ${fdt_addr_r}
```

Boot Using Static IP

To boot a kernel that doesn't use Device Tree:

```
usb start
setenv serverip <tftp_server_ip>
setenv ipaddr <a_spare_ip_address>
tftp ${kernel_addr_r} zImage
setenv bootargs earlyprintk console=ttyAMA0 console=tty1 root=/dev/mmcblk0p2 rootwait
bootz ${kernel_addr_r}
```

If your kernel uses Device Tree:

```
# swarren's branch already sets this automatically, so you can skip this
# Mainline U-Boot will set the following automatically soon
setenv fdtfile bcm2835-rpi-b.dtb

usb start
setenv serverip <tftp_server_ip>
setenv ipaddr <a_spare_ip_address>
tftp ${kernel_addr_r} zImage
tftp ${fdt_addr_r} ${fdtfile}
setenv bootargs earlyprintk console=ttyAMA0 console=tty1 root=/dev/mmcblk0p2 rootwait
bootz ${kernel_addr_r} - ${fdt_addr_r}
```

Needs specific values, or examples, for kernel_addr_r and fdt_addr_r. Procedure may work, but specifics are missing.

U-Boot script files

uEnv.txt

swarren's branch (and soon mainline) load file `/uEnv.txt` from the SD card prior to auto-booting and/or entering the command-line. This file can be used to customize the U-Boot environment at an early stage. Each line, of the format `name=value`, sets one environment variable.

boot.scr.uimg

swarren's branch and mainline both search all known storage devices for a file name `boot.scr.uimg` (or `boot.scr`) in either `/` or `/boot`. The SD card is searched, as is any USB storage device if USB support is enabled in U-Boot. The commands described in the various booting sections above may be placed into this file, and will be

automatically executed by U-Boot once the autoboot timer expires.

To create this file:

```
vi boot.scr
# Place the relevant set of boot commands into the file
mkimage -A arm -O linux -T script -C none -n boot.scr -d boot.scr boot.scr.uimg
```

Now, copy `boot.scr.uimg` to your SD card.

"DHCP" boot

swarren's branch (and mainline once USB support lands there) both attempt to download a file named `boot.scr.uimg` from a TFTP server. Create the file as described in the previous section, and place it in your TFTP server's data directory.

PXE

swarren's branch (and mainline once USB support lands there) both attempt to download and execute an extlinux/syslinux (BootLoaderSpec) config file from a TFTP server. Create e.g. `pxelinux.cfg/default` in the TFTP server's data directory. An interactive menu will be displayed for the various boot options described in this file.

boot_targets

The U-Boot environment variable `boot_targets` may be used to configure which auto-boot targets are executed at boot. uEnv.txt may be used to configure this variable.

- v
- t
- e

Raspberry Pi

Startup	Buying Guide - SD Card Setup - Basic Setup - Advanced Setup - Beginners Guide - Troubleshooting
Hardware	Hardware - Hardware History - Low-level peripherals - Expansion Boards
Peripherals	Screens - Cases - Other Peripherals (Keyboard, mouse, hub, wifi...)
Software	Software - Distributions - Kernel - Performance - Programming - VideoCore APIs - Utilities
Projects	Tutorials - Guides - Projects - Tasks - DataSheets - Education - Communities

RpiFront.jpg

Retrieved from "https://elinux.org/index.php?title=RPi_U-Boot&oldid=449171"

Categories:

- RaspberryPi
- U-Boot

Navigation menu

Personal tools

- Log in
- Request account

Namespaces

- Page
- Discussion

Variants

Views

- Read
- View source
- View history

More

Search

Search Go

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Help
- Volunteering
- Bug Tracker