The PowerPC boot wrapper

Copyright (C) Secret Lab Technologies Ltd.

PowerPC image targets compresses and wraps the kernel image (vmlinux) with a boot wrapper to make it usable by the system firmware. There is no standard PowerPC firmware interface, so the boot wrapper is designed to be adaptable for each kind of image that needs to be built.

The boot wrapper can be found in the arch/powerpc/boot/ directory. The Makefile in that directory has targets for all the available image types. The different image types are used to support all of the various firmware interfaces found on PowerPC platforms. OpenFirmware is the most commonly used firmware type on general purpose PowerPC systems from Apple, IBM and others. U-Boot is typically found on embedded PowerPC hardware, but there are a handful of other firmware implementations which are also popular. Each firmware interface requires a different image format.

The boot wrapper is built from the makefile in arch/powerpc/boot/Makefile and it uses the wrapper script (arch/powerpc/boot/wrapper) to generate target image. The details of the build system is discussed in the next section. Currently, the following image format targets exist:

cuImage.%:	Backwards compatible uImage for older version of U-Boot (for versions that don't understand the device tree). This image embeds a device tree blob inside the image. The boot wrapper, kernel and device tree are all embedded inside the U-Boot uImage file format with boot wrapper code that extracts data from the old bd_info structure and loads the data into the device tree before jumping into the kernel. Because of the series of #ifdefs found in the bd_info structure used in the old U-Boot interfaces, cuImages are platform specific. Each specific U-Boot platform has a different platform init file which populates the embedded device tree with data from the platform specific bd_info file. The platform specific cuImage platform init code can be found in *arch/powerpc/boot/cuboot.*.c*. Selection of the correct cuImage init code for a specific board can be found in the wrapper structure.
dtbImage.%:	Similar to zImage, except device tree blob is embedded inside the image instead of provided by firmware. The output image file can be either an elf file or a flat binary depending on the platform. dtbImages are used on systems which do not have an interface for passing a device tree directly. dtbImages are similar to simpleImages except that dtbImages have platform specific code for extracting data from the board firmware, but simpleImages do not talk to the firmware at all. PlayStation 3 support uses dtbImage. So do Embedded Planet boards using the PlanetCore firmware. Board specific initialization code is typically found in a file named arch/powerpc/boot/<platform>.c; but this can be overridden by the wrapper script.
simpleImage.%:	Firmware independent compressed image that does not depend on any particular firmware interface and embeds a device tree blob. This image is a flat binary that can be loaded to any location in RAM and jumped to. Firmware cannot pass any configuration data to the kernel with this image type and it depends entirely on the embedded device tree for all information.
treeImage.%;	Image format for used with OpenBIOS firmware found on some ppc4xx hardware. This image embeds a device tree blob inside the image.
uImage:	Native image format used by U-Boot. The uImage target does not add any boot code. It just wraps a compressed vmlinux in the uImage data structure. This image requires a version of U-Boot that is able to pass a device tree to the kernel at boot. If using an older version of U-Boot, then you need to use a cuImage instead.
zImage.%:	Image format which does not embed a device tree. Used by OpenFirmware and other firmware interfaces which are able to supply a device tree. This image expects firmware to provide the device tree at boot. Typically, if you have general purpose PowerPC hardware then you want this image format.

Image types which embed a device tree blob (simpleImage, dtbImage, treeImage, and cuImage) all generate the device tree blob from a file in the arch/powerpc/boot/dts/ directory. The Makefile selects the correct device tree source based on the name of the target. Therefore, if the kernel is built with 'make treeImage.walnut', then the build system will use arch/powerpc/boot/dts/walnut.dts to build treeImage.walnut.

Two special targets called 'zImage' and 'zImage.initrd' also exist. These targets build all the default images as selected by the kernel configuration. Default images are selected by the boot wrapper Makefile (arch/powerpc/boot/Makefile) by adding targets to the $image-y variable. Look at the Makefile to see which default image targets are available.

How it is built

arch/powerpc is designed to support multiplatform kernels, which means that a single vmlinux image can be booted on many different target boards. It also means that the boot wrapper must be able to wrap for many kinds of images on a single build. The design decision was made to not use any conditional compilation code (#ifdef, etc) in the boot wrapper source code. All of the boot wrapper pieces are buildable at any time regardless of the kernel configuration. Building all the wrapper bits on every kernel build also ensures that obscure parts of the wrapper are at the very least compile tested in a large variety of environments.

The wrapper is adapted for different image types at link time by linking in just the wrapper bits that are appropriate for the image type. The 'wrapper script' (found in arch/powerpc/boot/wrapper) is called by the Makefile and is responsible for selecting the correct wrapper bits for the image type. The arguments are well documented in the script's comment block, so they are not repeated here. However, it is worth mentioning that the script uses the -p (platform) argument as the main method of deciding which wrapper bits to compile in. Look for the large 'case "$platform" in' block in the middle of the script. This is also the place where platform specific fixups can be selected by changing the link order.

In particular, care should be taken when working with cuImages. cuImage wrapper bits are very board specific and care should be taken to make sure the target you are trying to build is supported by the wrapper bits.