Booting ZX parts

Broadcom Proprietary and Confidential

Release note: this document can be shared with customers.

bzx: booting [zd,zb] parts - android build adaptation.

CONTENTS

In our traditional build infrastructure, including build for google's reference devices, we make the assumption that the device is of a certain part type, which is typically set for the reference board to be a "ZD" (development) part.

In some rare but nonetheless valid use cases, we may however encounter situations where the part in question is not a ZD but a ZB (production) part. This can happen when shortage of designs forces the usage of non development targets, resulting in a possibility of a mix environment with both ZD and ZB parts available.

In this case, care must be taken with regards to a couple of things that the reference image does not automatically deal with but that is required for a ZD to ZB transition plan.

bootloader.img Adaptation for ZX Parts

Creating the Proper Bolt Image

Flashing a ZD bolt image on a ZB part (or vice versa in fact) would brick the device, recovery via bbs is necessary to flash a working bootloader (bolt.bin) image.

However it is simple enough to create a ZB compatible bootloader.img image from a ZD one (or vice-versa). Here is how in a few simple steps, assuming we started from a ZD part and need to adapt for a ZB part (you can adapt the other way around if necessary).

Prerequisits

You will need:

- A bootloader.img which is the bolt+bsu combination created by the android build system.
- **The bbl and bfw for the part of interest which can be found on this page**.
    - If you started from a ZD bootloader.img, grab the ZB firmware.

What To Do

Step by step, this is assuming you have built the bootloader.img and have access to such environment (the output of the build).

- split the bootloader.img from the build, using the provided script, this will produce a bolt.bin and a android_bsu.elf

```
./vendor/broadcom/bolt/android/scripts/split_bootloaderimg.py bootloader.img
```

- replace bbl firmware in the extracted **bolt.bin**. the **<target>** is the device you have built for. the zb-bbl-**<version>**.bin is the one from the page linked

```
./out/target/product/<target>/obj/FAKE/bolt/scripts/patcher.pl -p zb-bbl-
<version>.bin -i bolt.bin -o bolt.new.bin -z zeus42 -t bbl
```

- replace bfw firmware in the newly created **bolt.new.bin**. the **<target>** is the device you have built for. the zb-bfw-**<version>**.bin is the one from the page linked

```
./out/target/product/<target>/obj/FAKE/bolt/scripts/patcher.pl -p zb-bfw-
<version>.bin -i bolt.new.bin -o bolt.fin.bin -z zeus42 -t bfw
```

- repackage the new bootloader.img for flashing via fastboot

```
./vendor/broadcom/bolt/android/scripts/bootloaderimg.py bolt.fin.bin
android_bsu.elf bootloader.zbx.img
```

Sage Binaries for [ZD,ZB] in the Single Image

The sage binaries are also one aspect of the system that needs to be looked into carefully. Similarly to the bootloader integration, the default sage integration typically assumes a ZD part and includes only the development binaries.

However for the same image to work on a ZB part, one need to also package up the production binaries. Luckily this can be done using a device configuration profile export:

```
export LOCAL_DEVICE_SAGE_DEV_N_PROD := y
```

Add the above to your device/broadcom/<target>/<target>.mk device profile before building the image.

It should be noted that doing so will increase the size of the image due to the inclusion of sage binaries (sage bootloader, sage framework and sage TA's) for both parts. As such, this should never be done on a final production image as it would package up some useless modules which eat valuable space, but on a development target image, it may be acceptable.

Build Integration

The mechanism allowing override of the bfw|bbl binaries for chip specific configuration has been integrated with the image build environment. The bbl|bfw will now be automatically patched into a final bootloader.xxx.img if defined in the device configuration.

```
# bootloader firmware manipulation.
#
#   say that we want dual bootloader generation and dual sage firmware.
#
export LOCAL_DEVICE_SAGE_DEV_N_PROD := y
#
#   swap bbl with the define version for second bootloader image.
#
export BOLT_IMG_SWAP_BBL         := <device-path-from-top>/bbl-<version>-
<variant>.bin
#
#   swap bfw with the defined version for second bootloader image.
#
export BOLT_IMG_SWAP_BFW         := <device-path-from-top>/bfw-<version>-
<variant>.bin
```

The variables to be defined in the device tree settings are illustrated above, both defines are independent and can be changed together or separately.

The build will automatically produce the following bootloaders in this situation:

1. a "bootloader.dev.img" which is the default development device supported, which in most cases means a "zd" part.
2. a "bootloader.prod.img" which is only produced if the above configuration is provided and which in most cases means a "zb" part. In this secondary image, the bbl|bfw will be swapped using the defined ones.

Since this is purely a binary manipulation activity, there is no checks added to ensure the swap'ed in version are actually proper for the device, This is the responsibility of the user defining the override.

Example Configuration: elfin

The elfin design integration in the google code tree provides an example of the automatic integration of a zd|zb chip target. Elfin reference design is targeting zd chip internally, but majority of deployed devices in google hands are zb nowadays.

This is how such setup is made possible simply by properly enabling the variants in the "device/broadcom/elfin/common.mk":

```
# bootloader firmware manipulation and sage firmware dual inclusion.
export LOCAL_DEVICE_SAGE_DEV_N_PROD := y
export BOLT_IMG_SWAP_BBL            := device/broadcom/elfin/blb/zb/bbl-3.1.1-
zb.bin
export BOLT_IMG_SWAP_BFW            := device/broadcom/elfin/blb/zb/bfw-4.2.3-
zb.bin
```

With those few lines of configuration, the main image (system|vendor) is now defined to work on both zd|zb chip variant AND both a bootloader.dev.img (zd parts) and a bootloader.prod.img (zb parts) are being generated and packaged during build.