Guideline and Instructions

Note: *This demonstration was on a Jetson originally flashed with L4T 21.4 using JetPack 1.2 on the host PC*.

This is a demonstration of the simplest way to save the Jetson image. This technique will save what is called the 'APP' partition of the Jetson flash memory (which basically is the 'Root File System' and includes the working area of your system) to a file on the host PC which you used JetPack to flash the Jetson originally.

If you are a Linux expert and have added more partitions, or are working on partitions other than 'APP' on the system side, see the **Jetson/Cloning page on the Jetson wiki** for more information about cloning the Jetson. If you don't know what any of that gibberish means (like me), then you're in luck because this article is for you.

Cloning the Jetson is actually made up of two parts. The first part is saving the current 'image' from the Jetson flash memory as a file onto the JetPack PC. The second part is copying that image back to a Jetson. The Jetson may be the same one, or another Jetson TK1. As you might guess, this is useful when you are trying to replicate a Jetson on another Jetson, or multiple Jetsons.

Entering USB Recovery Mode

•Ensure a USB cable is connected from your host system to the USB micro B recovery connector on the board.
•Press and hold the "Force Recovery" button.
•If the board was off, apply power.
•If the board was on, press and release the reset button.
•Wait a short time (e.g. 1 second) and release "Force Recovery".

Saving the Image

In order to save the image, first you will need to navigate to the proper directory on the host PC to begin the process. Open a Terminal and 'cd' to the JetPack install directory.

For example, on a default install, the install directory is ~/JetPackTK1-1.2/

```
$ cd ~/JetPackTK1-1.2
```

Then, switch over to the boot loader directory:

```
$ cd Linux_for_Tegra/bootloader
```

At this point, you're almost ready to go. Make sure that the Jetson is attached to the PC using the supplied USB cable that you used to flash the Jetson with originally. Set the Jetson into recovery mode, as you did during the original flash. At this point, you should be able to execute on the PC:

```
$ lsusb
```

and see the Jetson which is identified as ***0955:7140 Nvidia Corp***.

To download the image from the Jetson, execute:

```
$ sudo ./nvflash --read APP clone.img --bl ardbeg/fastboot.bin --go
```

At this point, you should see magic happen. There will be a counter that is updated as the bytes are downloaded from the Jetson. There are 15GBs of data, so reading them takes a while over USB 2. On my machine it takes about 30 minutes. A PC system running from a virtual machine could take **much**longer.
Once the download is complete, you will see a file in the directory named ‘clone.img’ which is about 15 GBs in size. This is the downloaded image from the Jetson. You will want to change the permissions of clone.img:

```
$ sudo chmod 744 clone.img
```

You can move clone.img to another place to save it. ***Note:*** *the file can be compressed using Gzip to about a third the size, useful for archival purposes*.

Restoring the Image

Restoring a saved image consists of flashing the cloned image to a Jetson. This is the same procedure that happens ‘under the covers’ of JetPack except that instead of a basic ’empty’ system, everything that was loaded on the cloned device is flashed.
The file ‘system.img’ in the bootloader directory is the name of the file which represents the disk image of the Jetson. In order to use the cloned image, you will need to copy the file ‘clone.img’ to ‘system.img’. First, you will need to be in the boot loader directory as described before. For example:

```
$ cd ~/JetPackTK1-1.2/Linux_for_Tegra/bootloader
```

Copy the cloned image to the system image that will be flashed:

```
$ sudo cp clone.img system.img
```

Notes: *The clone.img file must be decompressed if you compressed it to save space. This also assumes that clone.img is still in the bootloader directory. If it was moved or the name changed, you will have to make the necessary adjustments. In any case, the uncompressed cloned image needs to end up in system.img in the****bootloader*** *directory.*
Next, make sure that the Jetson is connected to the PC through the micro connector USB cable as during normal flashing, and place the Jetson into recovery mode. At this point, you should be able to execute on the PC:

```
$ lsusb
```

and see the Jetson which is identified as ***0955:7140 Nvidia Corp***.
At this point, we’re ready to flash. Here’s the magic incantation:

```
$ cd ../

# As an example, you should be in ~/JetPackTK1-1.2/Linux_for_Tegra/

$ sudo ./flash.sh -r -S 14580MiB jetson-tk1 mmcblk0p1
```

The **-s** flag tells the flashing program to skip building and reuse the existing system.img file. The partition size (**-S 14580MiB**) is the default that JetPack uses.

More magic should happen as the Jetson is flashed. Note that it will take longer than normal to flash the Jetson in comparison to the empty system as it restores all 15GBs. In the demonstration, the process took about 15 minutes.
When the system is done flashing, reboot the Jetson. The Jetson will be restored to the state of the original at the time of cloning.

Discussion

During the course of developing on the Jetson it is likely that a large amount of time and effort went into loading libraries from repositories, compiling source code, adding applications, building and loading drivers, and configuration. Once a ‘work system’ is built, it can represent a substantial investment.
While almost all developers try to keep their source code updated in repositories such as Github, it seems inevitable that at some point the system represents more than its sum of parts. This is either because the system is out of synch with the external repositories, the smaller bits and pieces were over looked and not committed to the repository, or magical unicorns came through and rooted around with their horn and bonked the system.
Bonk is a technical term for when the system should work but doesn’t. This is prevalent when a project is due, or a demonstration is about to happen. You’ll find this happens most often when everything works, but an extra ‘feature’ is added at the last minute. In the rush to get the feature working, something bad happens, and … bonk.
Given enough time, people find that the cause of these problems is usually minor or a configuration issue. However, the results appear catastrophic at the time and under pressure the participants are often perplexed as to the cause.
Having a system snapshot gives the developer ‘something in their back pocket’ so that a system can be restored to a known working state, which can be invaluable during development.
Another advantage is that it makes it simple to build a system, and then flash it to other machines. This is useful for building clusters of machines, or building several prototypes at once.