Running Bamboo Build Scripts Locally

Chris Jenkins - Genesi USA, Inc

Terminology

- “Bamboo” refers to the GUI web interface and helper services provided by the Atlassian software suite for running a build server
- “Build scripts” refers to Genesi’s own software tools for building Efika MX53 images
- Launching a “Bamboo plan” only refers to launching one of the build scripts used by Bamboo
- “Debian”: Linux distribution dedicated to open source software, forming the basis of many other distros
- “Ubuntu”: Linux distribution based on Debian focused on user experience, in use on Drónov Slimbook
- “U-Boot”: Universal Bootloader, in use on Efika MX53 systems (such as Drónov Slimbook)

Prerequisites: GitHub

- Our current build scripts live in a private GitHub repository:

  https://github.com/genesi/bamboo-builders
- You will need:
  - Read permissions for repo (contact Genesi)
  - Git (prefered >= v.1.7.2)
  - To run locally clone with command:

    ```
    git clone https://github.com/genesi/bamboo-builders.git
    ```
  - Then switch to the correct branch:

    ```
    git checkout -t origin/docky-slimbook
    ```

Prerequisites: Software

- Your local machine should be based on Debian or (preferably) Ubuntu
- `git`: Open-source version control software
- `QEMU`: Open-source “QUick EMUlator”, used for building the Drónov Slimbook image when local machine is x86/AMD
- `binfmt-support`: local machine kernel support for e.g. associating files in the build with QEMU
- `u-boot-tools`: used to create boot scripts and bootable kernel images
- `debootstrap`: used to install Debian within a running Debian system.
- `parted`: advanced file-system partitioning software
- other misc. programs: `zerofree`, `pigz`, `pixz`, `pv`
    - README-LOCAL.MD will have more information for installing these

In particular, you will need a specific version of `dosfstools, 3.0.13 (30 Jun 2012)` available here

General Layout

- Bamboo plans and environments are emulated in bash scripts at the root of the repo.
- `maverick-build`, `maverick-dock` and `maverick-installer` are the only plans available locally.
  - Of these, `maverick-installer` and `maverick-dock` are of interest
- Each plan uses a common set of scripts in the repo root directory, as well as plan specific scripts in a directory of the same name in `installer/static`
  - e.g. executing `maverick-installer` will execute the code in `installer/static/maverick-installer`

Shared Build Scripts

- `00_setup_target`: create a blank image, setup partitions and write STM and U-Boot to boot partition, mount to loop device
- `01_debootstrap`: install Debian system on image
- `02_create_installer`: copies static files into place in image file system (source location depends on specific plan)
- `10_install`: runs script `installer/installer` (boots QEMU if not building on an ARM device, runs under `chroot` otherwise)
- `installer/installer`: executes build-specific scripts within new Debian system (under `QEMU` or `chroot`), then if QEMU power off
- `finalize`: unmounts target image, releases loop device

Build: Maverick Dock

- `10_create_files`: creates fstab, repo sources list, initramfs configs, stops startup services
- `20_packaging`: install Genesi / Ubuntu software from Genesi repository
- `30_cleanup`: creates user “oem”, prepares oem-config, creates build hostname, prepares kernel / U-Boot scripts for system boot, restores startup services

Build: Maverick Installer

- `10_create-files`: creates fstab, repo sources list, stops startup services
- `20_packaging`: install Genesi / Ubuntu software from Genesi repository, utility programs for installing target image
- `30_cleanup`: creates user “installer”, moves installer files to appropriate locations (system install script / root fs archive, U-Boot boot scripts), creates build hostname, prepares kernel / U-Boot scripts for system boot, restores startup services

Misc. Programs

- `local-setup`: code to fake Bamboo environment for local builds
- `functions`: functions doing most of the heavy lifting during the build
- `installer/static/${PLAN}/`
  - `etc/apt/*`: package manager configuration (sources list, preferences)
  - `etc/debconf/preseed.conf`: used to set default answers during system configuration by end user
  - `scripts/kernel-prep`: creates U-Boot executable kernel image / init ramdisk
  - `scripts/script-prep`: creates U-Boot executable boot script, after some string processing.

Running maverick-dock

- To build the root file system, execute the script `maverick-dock` without arguments
  - `./maverick-dock`
  - The script will then prompt you for your administrative password (sudo)
  - If the build finishes without error, it will generate two files in the root of the repo:
    - `maverick-dock-FAKE-JOB-000.img.gz` - SD card image
    - `maverick-dock-FAKE-JOB-000.tar.gz` - Root file system archive
- The build takes ~140 minutes on on our build server, can take >= 200 minutes on personal workstation

Running maverick-installer

- To build the installer image, execute the script maverick-install with a single argument, the path to the root file system archive generated by `maverick-dock`
  - `./maverick-installer maverick-dock-FAKE-JOB-000.tar.gz`
  - Again, it will prompt you for sudo password
  - If the build finishes without error, it will generate a file in the root of the repo:
    - `maverick-intaller-FAKE-JOB-000.img.gz` - SD card image
- The build takes ~30 minutes on our build server, can take >= 45 minutes on a personal workstation

Miscellaneous Advice

- Unless you have a very powerful personal workstation, the build process will consume considerable processing power. To minimize the memory and CPU used by other background processes, you may choose to launch your builds through the Linux CLI, accessible by Ctrl+Alt+F#, where F# is one of the numbered function buttons, F1 to F7
- There are many places an error can occur during a build, so we highly recommend running your builds in a `screen` session with logging enabled (in `screen` you can toggle logging with the key combination Ctr+a Shift+H)
- Should you wish to extend the Bamboo build scripts, note that a given set of scripts are always executed in order, e.g. `00_setup_target` is executed before `01_debootstrap` because 00 comes before 01. Scripts living in differing directories are never compared in this way.

After the Build: Creating an Installer SD and the Install Process

Creating an Installer SD

- Insert an SD card with at least 4GiB storage in your workstation
- Make sure the SD card is not mounted
  - ```
    $ mount | grep mmc
    /dev/mmcblk0p3 on /media/user/MMCSystem type ext4 (rw,nosuid,nodev,uhelper=udisks2)
    /dev/mmcblk0p1 on /media/user/MMCBOOT type vfat (rw,nosuid,nodev,uid=1000,gid=1000,
    shortname=mixed,dmask=0077,utf8=1,
    showexec,flush,uhelper=udisks2)
    ```
  - `$ sudo umount /dev/mmcblk0p?`
- Each Bamboo plan creates a `.img.gz` file, which is a compressed disk image. Uncompress and write your SD card:
  - `zcat ${PATH_TO}/${BAMBOO_PLAN}.img.gz | sudo dd of=/dev/mmcblk0 bs=32M`
  - This says, “Pass the uncompressed image data to `dd` and have it write directly to the SD in large chunks (fewer disk operations -> faster writes)
  - If you want a nice progress bar, make sure `pv` is installed and use this command:

    `zcat ${PATH_TO}/${BAMBOO_PLAN}.img.gz | pv --size=3800m | sudo dd of=/dev/mmcblk0 bs=32M`
    ```

SD Installer

- On first boot the SD installer will execute `/home/installer/target_installer`
- This script makes system checks (board, privileges, boot and root NAND partitions are mounted), gets system time and asks the user to continue
- If user confirms, uses programs from `mtd-tools` to flash the nand boot and root partitions, attach partitions to ubi file system, and makes boot and root ubifs volumes
- Extract the root file system archive to the new NAND root file system
    - This includes uImage and uInitrd
- Cleanup
    - Make sure `oem-config` runs on first boot
    - Create unique, temporary hostname
    - Copy `boot.ubi.scr` on SD to boot partition
    - Convert SD from installer to configurable boot SD

Aside: SD Partitions

For various reasons, our partition layout is rather advanced.

```
$ sudo parted -l /dev/mmcblk0
Model: SD SD08G (sd/mmc)
Disk /dev/mmcblk0: 7986MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End    Size    Type     File system  Flags
 2      1024B   4194kB  4193kB  primary               boot, diag
 1      4194kB  134MB   130MB   primary  fat16
 3      134MB   3773MB  3639MB  primary  ext4
```

- First physical partition / second logical partition is for bootloader. (loader.bin and U-Boot live there)
- Second physical partition / first logical partition is Linux boot (uImage and uInitrd live there)
- Third partition is Linux root file system

genesi