**LƯU AN PHÚ**

Uboot

Architecture and Porting

Facebook group: Cùng Nhau Học Linux Kernel

MỤC LỤC

[CHƯƠNG 1. Flow source code của uboot 2](#_Toc63935926)

[1.1 Mở đầu 2](#_Toc63935927)

# Flow source code của uboot

## Mở đầu

Do hệ điều hành có thể nằm trên các vùng nhớ khác nhau như HDD, flash, mmc, sdcard, thậm chí có thể nằm trên internet. Do vậy khi máy tính khởi động, nó cần chạy 1 chương trình đặc biệt dùng để load hệ điều hành. Chương trình đó gọi là boot loader. Nếu như hệ điều hành có thể lưu trữ tại bất cứ đâu, có thể là bộ nhớ nằm trong máy tính hoặc nằm ngoài máy tính (lưu trữ trên internet) thì boot loader thông thường chỉ lưu trữ trên ROM (Read Only Memory). Khi power on, CPU sẽ tự load boot loader và thực hiện câu lệnh đầu tiên của nó. Boot loader sau đó sẽ khởi tạo các tài nguyên khác của hệ thống như CPU, Ram, ethernet, sau đó nó sẽ tiến hành load hệ điều hành.

Boot loader là tên chung định danh cho loại chương trình được chạy trước hệ điều hành và có nhiệm vụ là load hệ điều hành. Trong thực tế boot loader có rất nhiều loại như Grub dùng cho PC, Uboot dùng cho các thiết bị embedded Linux.

Các công ty khi làm product chạy hệ điều hành Linux hoặc Android thường sẽ phát sinh nhu cầu chỉnh sửa code của uboot do mạch đã được làm lại khác với sample board. Ví dụ như thay đổi port serial mặc định, thay đổi nơi lưu trữ OS từ MMC sang sdcard… Trong tài liệu này mình sẽ trình bày kiến trúc source code của Uboot và cách porting nó cho board mới.

Trong tài liệu này mình sẽ đưa ra sample code và thực hành trên phiên bản Uboot mới nhất và chạy trên board Beagle Bone Black. Các bạn có thể tải source code và làm theo chỉ dẫn của hãng. Đây là link hướng dẫn:

https://www.digikey.com/eewiki/display/linuxonarm/BeagleBone+Black

## Uboot basic

Về bản chất thì Uboot là 1 chương trình vi điều khiển. Nó sử dụng trực tiếp địa chỉ vật lý chứ không thông qua virtual memory như Linux. Quá trình chạy của boot loader được chia làm 3 giai đoạn như sau:

### First stage boot loader.

Khi vừa bật nguồn thì CPU sẽ ngay lập tức thực thi 1 chương trình nhỏ được lưu trữ trong ROM. Quá trình CPU nhảy tới thực hiện câu lệnh tại địa chỉ cố định trong ROM được thực thi tự động bởi hardware trong chip và cố định đối với từng loại kiến trúc như Arm, intel… Luồng thực thi của chương trình nằm trong ROM này được gọi là first stage boot loader. Nhiệm vụ của first stage nhằm khởi tạo những tài nguyên tối thiểu như 1 core của CPU, static ram, sdcard (nếu như uboot được lưu trữ trên sdcard). Do chương trình first stage boot loader thông thường được viết bởi hang cung cấp chip nên trong tài liệu này mình sẽ không đi sâu.

### Second stage boot loader.

Sau khi first stage thực thi xong, nó sẽ load tiếp chương trình thứ 2 là second stage boot loader được lưu trên scard của beage bone black. Sau khi build xong uboot của beagle bone, chúng ta sẽ có 2 file MLO và uboot.img như hình dưới đây:

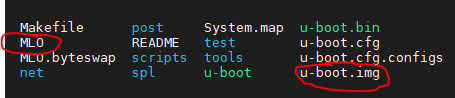


Figure 1 Các file sau khi build uboot

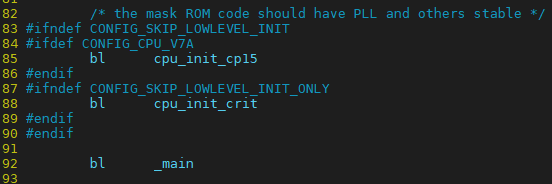
Đây là 2 câu lệnh liên tiếp dùng để ghi uboot vào thẻ nhớ trên board beage bone:

*sudo dd if=./u-boot/MLO of=${DISK} count=1 seek=1 bs=128k*

*sudo dd if=./u-boot/u-boot.img of=${DISK} count=2 seek=1 bs=384k*

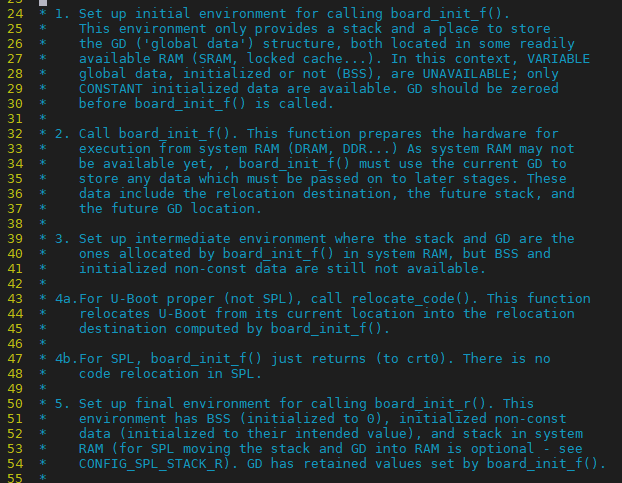
File chương trình second stage (MLO) được ghi vào block đầu tiên của thẻ sdcard và third stage (u-boot.img) được ghi vào block thứ 2 có địa bắt đầu là 128k. Như vậy sau khi boot rom chạy xong nó sẽ nhảy đến câu lệnh đầu tiên được lưu trữ trên thẻ nhớ và đó chính là câu lệnh đầu tiên của MLO (second stage). Chúng ta sẽ cùng tìm hiểu chi tiết các công việc mà second state thực hiện.

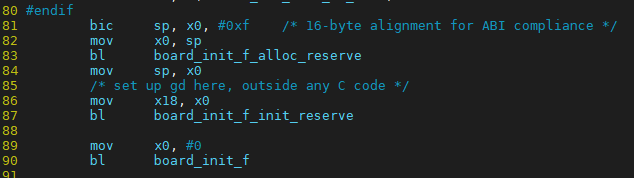
Những câu lệnh đầu tiên mà second stage thực hiện sẽ nằm trong file *arch/arm/cpu/armv7/start.S.* Trong source code u-boot có rất nhiều file start.S, mỗi một loại chip sẽ sử dụng 1 file start.S khác nhau. File này do nhà sản xuất chip như Arm hoặc Intel viết riêng cho chip của họ. Đối với Beagle bone sử dụng armv7 nên file start.S của nó sẽ có đường dẫn như trên. Dưới đây là 1 đoạn code nhỏ của start.S:

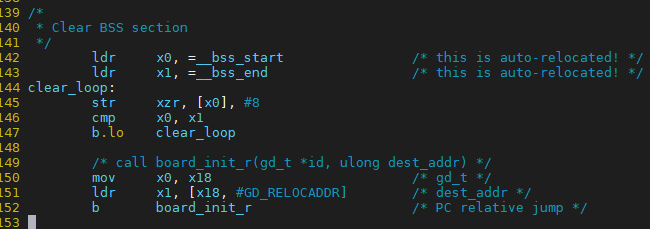


Vì chưa được khởi tạo tài nguyên như memory nên start.S thực thi với rất nhiều giới hạn. Nó không truy cập vào bộ nhớ, không sử dụng variable. Thay vào đó nó chỉ thao tác trên các thanh ghi mà thôi. Start.S làm các công việc rất cơ bản như khởi tạo cpu ở mức basic… Nhiệm vụ của nó là chuẩn bị tài nguyên để thực thi hàm \_main.

Hàm \_main nằm trong file *arch/arm/lib/crt0\_64.S.* Code của nó như sau:

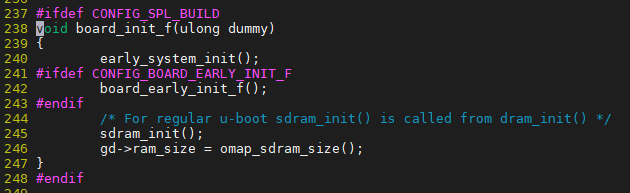




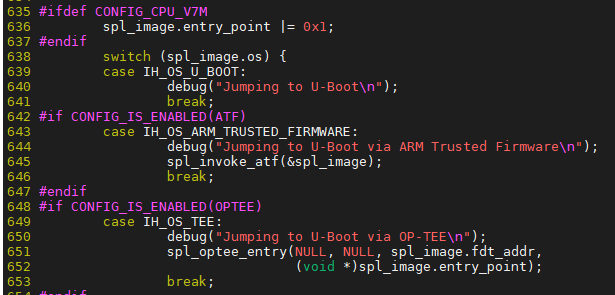


Nhiệm vụ của hàm \_main là khởi tạo tiếp các tài nguyên cần thiết như memory rồi gọi hàm board\_init\_f. Sau đó nó sẽ gọi hàm board\_init\_r.

Hàm board\_init\_f của second stage có source code nằm ở file *arch/arm/mach-omap2/hwinit-common.c.* Source code của file start.S do các hãng chip viết, còn source code của board\_init\_f do các hãng làm SoC (system on chip) viết. Do cả 2 được build cùng vào 1 chương trình là MLO nên chúng có thể gọi được nhau. Sau đây là source code của hàm board\_init\_f trong first stage:



Board\_init\_f sẽ khởi tạo sdram chính là bộ nhớ cached L2 của chip. Ngoài ra nó có thể khởi tạo 1 số tài nguyên quan trọng khác. Sau khi board\_init\_f thực hiện xong, \_main sẽ gọi tiếp hàm board\_init\_r. Board\_init\_r của second stage có source code nằm ở file: *common/spl/spl.c*

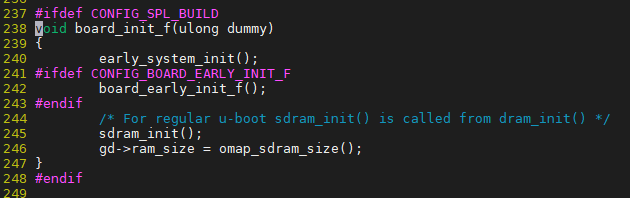


Board\_init\_r sẽ khởi tạo thêm 1 số thứ nữa, cuối cùng nó sẽ load image của chương trình third stage boot loader (chính là file u-boot.img) vào sdram. Tham số spl\_image trong hình chính là struct chứa thông tin của file MLO như start\_address, end\_address trên sdcard. Như mình đã nói ở trên thì u-boot.img sẽ có địa chỉ trên sdcard ngay sau end\_address của MLO.

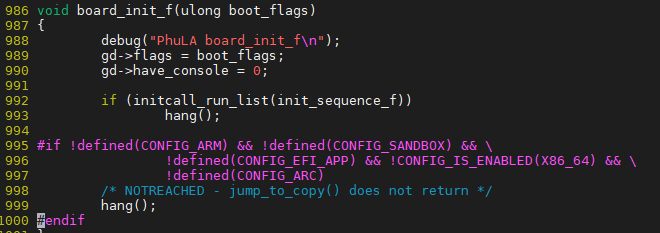
### Third stage boot loader

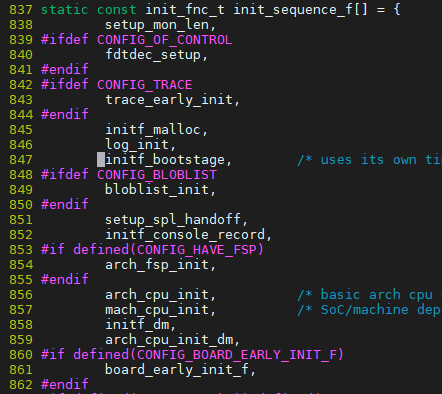
Sau khi u-boot.img được load vào dram, nó sẽ chiếm toàn quyền sử dụng hệ thống. Trong hệ thống không còn sự tồn tại của chương trình second state MLO nữa. So với second stage thì chương trình third stage sẽ làm được nhiều việc hơn do hệ thống đã khởi tạo được nhiều thứ hơn như đã có dram, có thể sử dụng hàm malloc,…

u-boot.img sẽ gọi hàm board\_init\_f và board\_init\_r trong source code của nó. Mỗi hàm này đều có 1 cặp, trong đó 1 cái sẽ nằm trong MLO và cái còn lại sẽ nằm trong u-boot.img. Hàm nằm trong u-boot.img sẽ làm được nhiều việc hơn so với MLO. Các hàm được build cùng với chương trình first stage sẽ có chỉ thị tiền xử lý CONFIG\_SPL\_BUILD ở đầu. Giống như hình bên dưới:



Hàm board\_init\_f của third stage có source code nằm ở file *common/board\_f.c.*

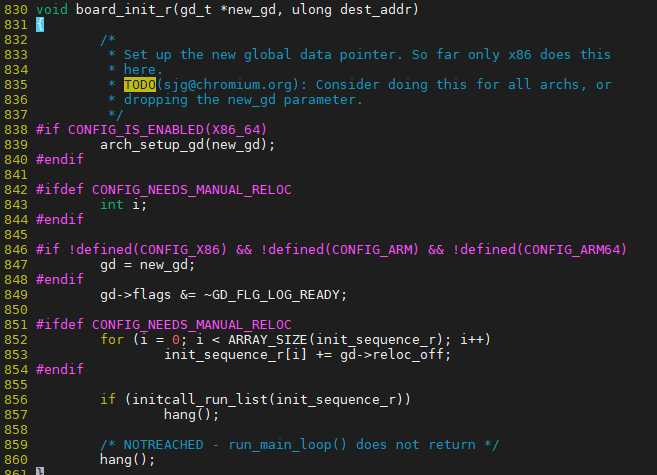


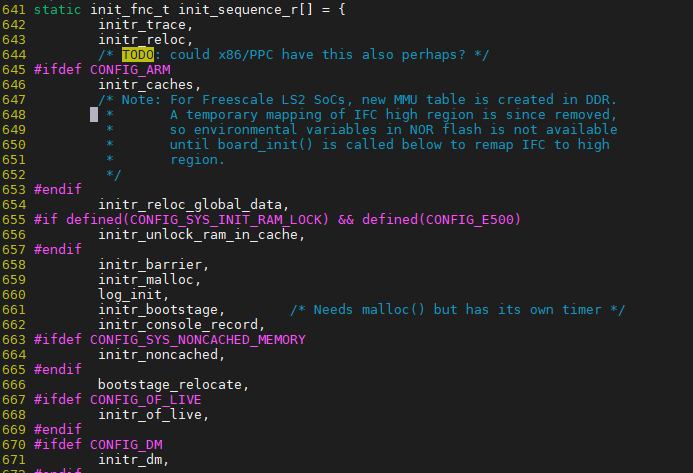


Nhiệm vụ của board\_init\_f là khởi tạo các tài nguyên dựa trên tài nguyên bộ nhớ ít ỏi là sdram ram. Luồng gọi sẽ như sau: board\_init\_f -> init\_sequence\_f -> khởi tạo từng module. Thông thường các hàm khởi tạo hardware sẽ gọi xuống driver của từng hãng khác nhau. Ví dụ như đối với beagle bone: init\_sequence\_f -> serial\_init -> probe (*drivers/serial/serial\_omap.c*).

Sau khi board\_init\_f chạy xong, lúc này hệ thống đã khởi tạo được nhiều tài nguyên hơn. Ví dụ như đã sử dụng được DRAM, dùng được malloc, device tree…

Lúc này \_main tiếp tục gọi board\_init\_r để khởi tạo nốt những module hardware còn lại. Tương tự như trên thì board\_init\_r cũng gọi xuống các file driver của board beagle bone theo luồng như sau board\_init\_r -> init\_sequence\_r -> khởi tạo từng module:





Sau khi khởi tạo xong hết hardware, board\_init\_r sẽ gọi ra hàm run\_main\_loop. Hàm này sẽ khởi tạo hệ thống command line của u-boot. Vậy command line trên uboot dùng để làm gì? Chúng ta sẽ cùng tìm hiểu ở phần tiếp theo.

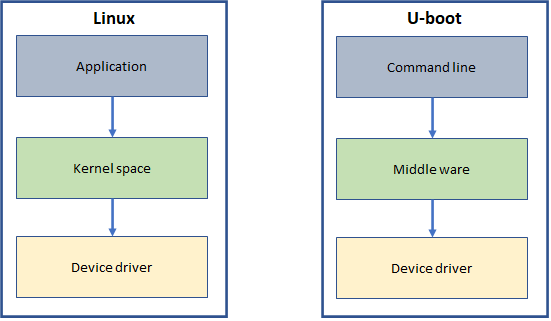
*Lưu ý:*

*1. 1 số tài liệu trên mạng cho rằng hàm board\_init\_f trong file common/board\_f.c và board\_init\_r trong file common/board\_r.c thuộc về luồng second stage (MLO) tuy nhiên mình không nghĩ như vậy. Do source code của 2 hàm này khá dài không phù hợp với kích thước của MLO, ngoài ra trong hàm board\_init\_f có khởi tạo device tree mà device tree là data được đính kèm ở cuối file u-boot.img. Do vậy mình kết luận rằng chúng thuộc về third stage.*

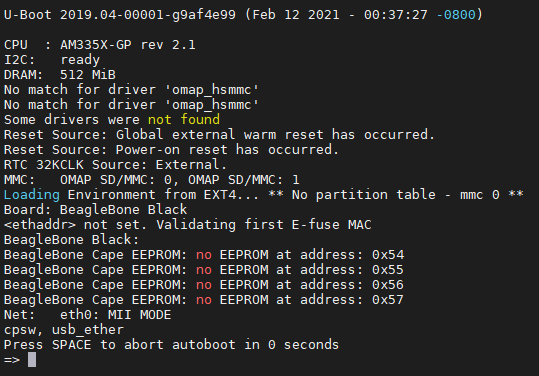
*2. u-boot khởi tạo dram thông qua luồng init\_sequence\_f -> dram\_init. Trong dram\_init sẽ get ra size của bộ nhớ ram trong hệ thống rồi truyền giá trị đó cho Linux thông qua kernel parameter. Nếu sau khi các bạn tích hợp thêm ram vào board mà hệ thống nhận không đủ ram thì có thể kiểm tra trong hàm này.*

### Hệ thống command line của u-boot

Nếu như Linux được phân chia làm 3 tầng bao gồm application, kernel, driver thì u-boot cũng tương tự như vậy.



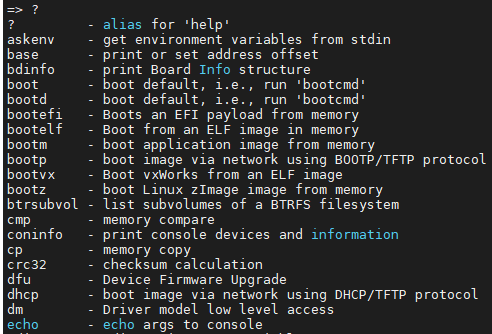
Sau khi khởi tạo hardware và middle ware thông qua 2 hàm board\_init\_f và board\_init\_r thì u-uboot sẽ đi vào 1 vòng lặp. Lúc này người dùng có thể thao tác gõ command line để điều khiển hệ thống. Muốn vào được giao diện này thì chúng ta phải connect PC vào board qua cổng serial và ngắt quá trình auto boot của hệ thống bằng phím space. Giao diện thao tác command line sẽ giống như hình sau:





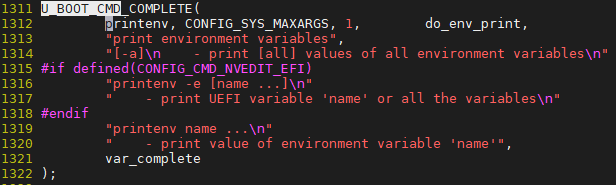
Sau khi ghi u-boot mới build được vào thẻ nhớ, các bạn lưu ý ấn phím s2 trên board để chương trình boot-rom jump sang uboot ở sdcard. Nếu không board sẽ vẫn sử dụng u-boot cũ trên eMMC. Mỗi lần build thì thời gian của hệ thống build cũng được đính kèm vào u-boot.img nên các bạn có thể thấy trên hình ở dòng U-Boot 2019.04-00001-g9af4e99 (Feb 12 2021 - 00:37:27 -0800).

U-boot support hàng trăm command line khác nhau như hình dưới đây:



Ngoài ra u-boot cũng có hệ thống biến môi trường giống như Linux. Để show ra giá trị các biến môi trường, các bạn có thể dùng câu lệnh printenv giống Linux.

Trên Linux thì các command line bản chất là các file binary được đặt trong thư mục /bin hoặc /sbin của hệ thống. Tuy nhiên do u-boot là một chương trình vi điều khiển, do vậy các command line của nó là các function trong source code. Mỗi 1 function khi muốn đăng ký thành command line sẽ sử dụng macro U\_BOOT\_CMD giống như hình dưới:



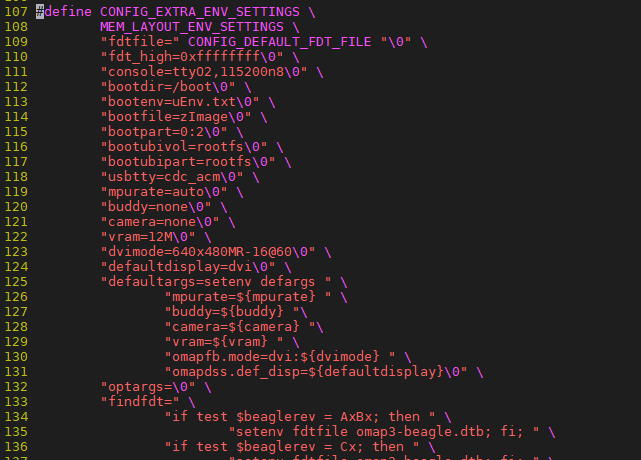
Command line của u-boot cũng hỗ trợ help và truyền option khi sử dụng giống như Linux. Hàm do\_main\_loop sẽ liên tục get input của user thông qua serial, từ đó detect được command line mà user gõ. Ví dụ như user gõ printenv thì hàm do\_env\_print sẽ được gọi ra để xử lý. Tất cả các command line còn lại đều tương tự như vậy. Một số comman line như đọc data trên mmc, i2c thì sẽ gọi xuống middle ware rồi từ middle ware sẽ call driver của board để thực hiện.

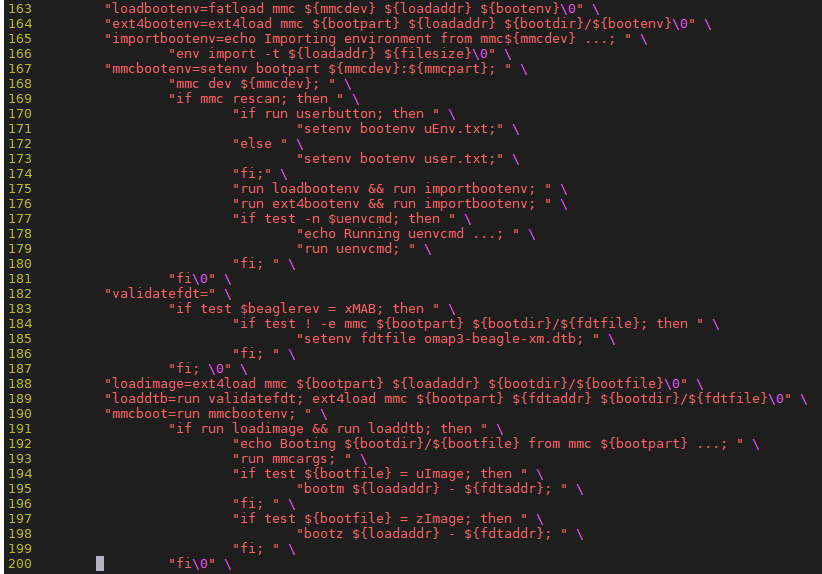
Nếu như quá trình boot không bị interrupt bởi input từ user thì hàm do\_main\_loop sẽ thực thi 1 loạt command line để execute Linux kernel. Quá trình này giống như chạy shell script. Chúng ta sẽ cùng tìm hiểu kỹ hơn ở phần tiếp theo.

### Quá trình load Linux kernel

Để load được image của Linux kernel lên ram thì cần thực thi 1 chuỗi các command line. Mỗi 1 board lại khác nhau đôi chút về các command line cần phải thực thi. Ví dụ như đối với beagle bone black thì cần load file uEnv.txt trên sdcard trước để lấy config hệ thống. Do vậy mỗi 1 hãng làm board sẽ tạo 1 script riêng và u-boot sẽ thực thi script đấy để load kernel image mà không cần quan tâm bên trong là gì.

Script đấy được define bằng macro CONFIG\_EXTRA\_ENV\_SETTINGS nằm trong folder /include/configs. Ví dụ như beagle bone sẽ có file include/configs/omap3\_beagle.h của riêng nó. Giải thích một chút về script boot kernel image của beagle bone như sau:





Đầu tiên u-boot sẽ đọc file /boot/uEnv.txt để lấy config. Sau đó sẽ lựa chọn load kernel image và dtb file từ mmc hoặc sdcard. Sau khi đọc kernel image và dtb file từ sdcard lên ram uboot sẽ dựa vào định dạng của kernel image và uImage hoặc zImage để tiến hành giải nén. Cuối cùng uboot sẽ gọi command bootm và truyền vào đó địa chỉ của kernel image và dtb file trên ram. Bootm sẽ jump vào câu lệnh đầu tiên của kernel và chuyển toàn bộ quyền điều khiển hệ thống cho nó. Đến đây là u-boot đã hoàn thành nhiệm vụ của mình.

### Tổ chức source code của u-boot.

Khi làm việc với u-boot, người lập trình viên phải biết tính năng A có source code nằm ở folder nào. Từ đó thu hẹp phạm vi tìm kiếm. Về cơ bản thì u-boot có tổ chức source code tương tự kernel. Bao gồm những folder chính sau:

1. cmd: Chứa source code của tất cả các command line. Thông thường mỗi command line sẽ là 1 file source C.

2. configs: Chứa file config để generate ra file .config khi build u-boot.

3. Documentation: Chứa hệ thống tài liệu của u-boot.

4. env: Chứa source code để xây dựng ra hệ thống biến môi trương của u-boot.

5. lib: Source code phần middler ware của u-boot.

6. net: Source code liên quan đến tính năng network. U-boot cũng có thể sử dụng được network như ping, sftp,…

7. arch: Source code specific cho từng platform. Device tree cũng được lưu trữ trong folder này.

8. common: Source code phần middler ware của u-boot.

9. drivers: Chứa source code driver của từng board.

10. spl: Chứa 1 phần code để build ra second stage boot loader.

11. tools: Chứa các loại tool dùng trong quá trình build u-boot.

12. fs: Chứa source code của các loại file system dùng trong u-boot. U-boot cũng có tính năng đọc file theo file name như Linux.