**Build and Boot U-Boot and Linux on a Raspberry Pi 3 Model B+**

[Linux](https://www.thegoodpenguin.co.uk/blog/category/tech-blog/linux/) [Tech Blog](https://www.thegoodpenguin.co.uk/blog/category/tech-blog/)

With more than [30 million](https://twitter.com/EbenUpton/status/1205646606504275968) units in existence, the [Raspberry Pi](https://www.raspberrypi.org/) provides for an accessible and low-cost way to play with an [ARMv8](https://en.wikipedia.org/wiki/ARM_architecture#64/32-bit_architecture) embedded device using the latest and greatest upstream sources. In this post we’ll walk through the steps required to build U-Boot, the Linux kernel and a filesystem, as well as the steps required for preparing an SD card and booting it.

Let’s start by getting a toolchain that will build 64bit ARMv8 binaries, we’ll grab the latest [Linaro toolchain](https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu/), unpack it and temporarily put it in our system path:

$ wget https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu/gcc-linaro-7.5.0-2019.12-x86\_64\_aarch64-linux-gnu.tar.xz

$ tar -xf gcc-linaro-7.5.0-2019.12-x86\_64\_aarch64-linux-gnu.tar.xz -C ~/tools

$ export PATH=~/tools/gcc-linaro-7.5.0-2019.12-x86\_64\_aarch64-linux-gnu/bin:$PATH

Even though the Raspberry Pi can [directly boot a kernel](https://www.raspberrypi.org/documentation/configuration/config-txt/boot.md), we’d still like to boot via U-Boot. Thus let’s grab the latest version of mainline U-Boot, configure it for our Raspberry Pi and build a u-boot.bin binary:

$ git clone ~~git://git.denx.de/u-boot.git~~ https://source.denx.de/u-boot/u-boot.git # v2021.01-rc2-121-g5b8991c667f7

$ cd u-boot

$ make ARCH=arm CROSS\_COMPILE=aarch64-linux-gnu- rpi\_arm64\_defconfig

$ make ARCH=arm CROSS\_COMPILE=aarch64-linux-gnu- -j$(nproc)

$ cd ..

make ARCH=arm CROSS\_COMPILE=aarch64-linux-gnu- -j$(nproc) để tạo ra u-boot.bin

make menuconfig để chỉnh lại các config (support ethernet)

Máy lab 25 thiếu package : sudo apt-get install srecord

**Initramfs là gì ?**

Trong quá trình khởi động một hệ thống Linux qua mạng, root filesystem (rootfs) là một phần cần thiết cho hình ảnh kernel để hoạt động. Rootfs có thể được định nghĩa theo một số cách, và một trong số đó là sử dụng một initial RAM disk (initrd) hoặc initramfs.

[**Initial RAM Disk (initrd)** là một hệ thống file tạm thời được tải vào bộ nhớ RAM trước khi hệ thống file gốc thực sự được mount1](https://en.wikipedia.org/wiki/Initial_ramdisk). [Initrd chứa các driver và module cần thiết để kernel có thể truy cập vào hệ thống file gốc và các thiết bị lưu trữ2](https://www.kernel.org/doc/html/latest/admin-guide/initrd.html). [Điều này đặc biệt quan trọng khi hệ thống file gốc nằm trên các thiết bị hoặc trong các cấu hình đặc biệt như RAID, LVM, hoặc NFS1](https://en.wikipedia.org/wiki/Initial_ramdisk).

[**Initramfs** là một phiên bản cải tiến của initrd, là một hệ thống file gốc tạm thời được lưu trữ dưới dạng một lưu trữ cpio nén và được tải vào bộ nhớ RAM trong quá trình khởi động3](https://ngelinux.com/difference-between-initrd-and-initramfs-all-what-you-need-to-know/). [Initramfs thường được sử dụng để thực hiện các công việc chuẩn bị trước khi mount hệ thống file gốc thực sự](https://en.wikipedia.org/wiki/Initial_ramdisk)[3](https://ngelinux.com/difference-between-initrd-and-initramfs-all-what-you-need-to-know/).

Để tạo một file initramfs, bạn có thể sử dụng công cụ như dracut trên các hệ thống Linux:

dracut /boot/initramfs-$(uname -r).img $(uname -r)

[Lệnh trên sẽ tạo một file initramfs mới cho phiên bản kernel hiện tại và lưu nó trong thư mục /boot4](https://linuxconfig.org/how-to-build-an-initramfs-using-dracut-on-linux).

Trong một file cấu hình của bootloader, bạn có thể sử dụng lệnh INITRD để chỉ định file initramfs hoặc initrd sẽ được sử dụng khi khởi động:

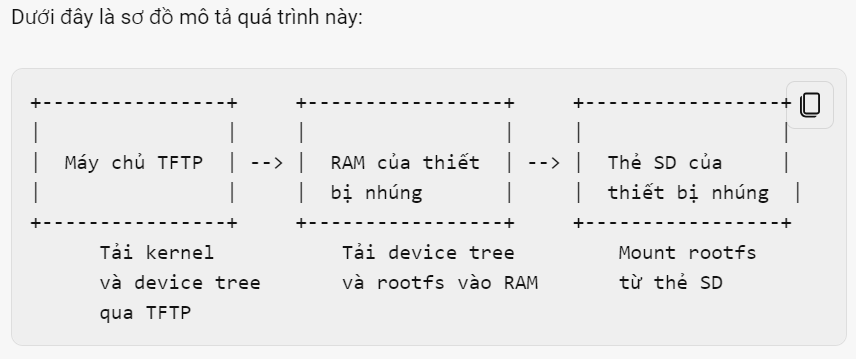
INITRD /boot/initramfs-$(uname -r).img

[Khi hệ thống khởi động, bootloader sẽ tải kernel và initramfs vào bộ nhớ, sau đó kernel sẽ giải nén initramfs và sử dụng nó như một hệ thống file gốc tạm thời để tiếp tục quá trình khởi động và cuối cùng mount hệ thống file gốc thực sự từ một thiết bị lưu trữ2](https://www.kernel.org/doc/html/latest/admin-guide/initrd.html).

**Giới thiệu về TFTP boot sử dụng u-boot và lưu rootfs trên sd card (lưu ý kernel không được lưu lại, mỗi lần chạy lại phải load kernel vào bằng u-boot – mục đích để thử nghiệm kernel, phát triển driver)**

Trong quá trình khởi động qua TFTP, kernel cũng được tải vào RAM, cùng với device tree và rootfs. Dưới đây là quy trình đầy đủ và chính xác:

1. **Nạp Bootloader**: U-Boot, là bootloader, được nạp từ bộ nhớ không bay hơi (như flash) và chạy đầu tiên khi thiết bị khởi động.
2. [**Nạp Kernel qua TFTP**: U-Boot sử dụng lệnh tftpboot để tải kernel từ máy chủ TFTP vào một địa chỉ cố định trong RAM1](https://wiki.st.com/stm32mpu/wiki/How_to_boot_the_kernel_via_TFTP_from_U-Boot).
3. [**Nạp Device Tree qua TFTP**: Device tree (DTB) cũng được tải vào RAM thông qua TFTP, thường là vào một địa chỉ sau kernel1](https://wiki.st.com/stm32mpu/wiki/How_to_boot_the_kernel_via_TFTP_from_U-Boot).
4. **Khởi Động Kernel**: U-Boot chuyển quyền điều khiển cho kernel bằng cách truyền địa chỉ của kernel và device tree trong RAM.
5. **Mount Rootfs**: Kernel sẽ khởi động và mount rootfs từ thẻ SD nếu đã được chỉ định trong bootargs, hoặc từ NFS nếu được cấu hình như vậy.



Từ đây ta có thể kiểm tra như bình thường

Nếu muốn chuyển kernel vào trong sdcard luôn thì đứng trong terminal linux thực hiện

sudo dd if=u-boot.bin of=/dev/sdx bs=512 seek=2

Trong đó:

* if=u-boot.bin là đường dẫn đến file kernel trong RAM.
* of=/dev/sdx là đường dẫn đến thiết bị SD card, với x là ký tự đại diện cho thiết bị SD card của bạn.
* bs=512 là kích thước block, 512 bytes là thông dụng.
* seek=2 là số block cần bỏ qua trước khi bắt đầu ghi, để tránh ghi đè lên MBR hoặc bất kỳ dữ liệu quan trọng nào khác.

**Lưu ý**: Bạn cần thay thế /dev/sdx bằng đường dẫn thực tế đến thiết bị SD card của bạn và u-boot.bin bằng đường dẫn đến file kernel trong RAM.

Hoặc sử dụng [mmc write tại u-boot](https://stackoverflow.com/questions/21455450/copying-kernel-and-uboot-into-sdcard) để lưu vào sdcard

**Giới thiệu về TFTP boot, roofs on NFS, no DHCP (rootfs nằm trên máy host)**

Dưới đây là một ví dụ về quy trình khởi động qua TFTP với root filesystem (rootfs) trên NFS mà không sử dụng DHCP, cùng với sơ đồ và giải thích cách hoạt động:

1. **Cấu hình IP tĩnh**: Trước tiên, bạn cần cấu hình một địa chỉ IP tĩnh cho thiết bị nhúng của bạn trong U-Boot. Điều này đảm bảo rằng thiết bị sẽ luôn có cùng địa chỉ IP mỗi khi khởi động và không phụ thuộc vào DHCP.
2. **Tải Kernel và Device Tree qua TFTP**: U-Boot sẽ tải kernel và device tree từ máy chủ TFTP vào RAM của thiết bị nhúng.
3. **Thực Hiện**bootcmd: U-Boot sẽ thực hiện lệnh trong biến môi trường bootcmd, bao gồm việc tải kernel và device tree từ máy chủ TFTP vào RAM.
4. **Cấu hình Bootargs**: U-Boot cần được cấu hình để truyền bootargs cho kernel, bao gồm thông tin về việc mount rootfs từ NFS.
5. **Khởi Động Kernel**: U-Boot chuyển quyền điều khiển cho kernel, kernel sau đó sẽ sử dụng thông tin trong bootargs để kết nối với máy chủ NFS và mount rootfs.
6. A screenshot of a computer code

   Description automatically generated**Hệ Thống Hoạt Động**: Kernel sẽ khởi động hệ thống với rootfs được mount từ NFS, cho phép bạn thực hiện thay đổi và cập nhật rootfs mà không cần phải ghi lại dữ liệu trên thiết bị nhúng.

**(lưu ý ko tải rootfs vào RAM)**

TFTP là để truyền file (kernel, device tree) và u-boot sẽ đứng ra sử dụng TFTP protocol và pxe command để truyền và ghi kernel và device tree vào ram (đã được mmc từ trước). Lúc này boot arguments sẽ set đường dẫn tới rootfs (chứa ở NFS server). Sau khi kernel được load thì lúc này tftp đã hết nhiệm vụ, kernel lúc này sẽ sử dụng thông tin từ boot argument ở trước đó để kết nối NFS và mount vào virtual filesystem interface. Từ điểm này trở đi, mọi truy cập file cần thiết cho việc hoạt động của hệ thống sẽ thông qua kết nối NFS, và TFTP không còn được sử dụng nữa.

**More information:**

NFS, viết tắt của Network File System, là một giao thức mạng được sử dụng để chia sẻ file giữa các máy tính trên một mạng. [NFS cho phép một máy tính (client) truy cập vào file và thư mục trên một máy khác (server) như thể chúng được lưu trữ trực tiếp trên ổ cứng cục bộ của client1](https://vietnix.vn/nfs-la-gi/).

**Cách hoạt động của NFS**:

1. **Máy chủ NFS**: Máy chủ này cấu hình và chia sẻ một hoặc nhiều thư mục của mình. Nó quản lý các yêu cầu từ các máy khách và cung cấp quyền truy cập vào file dựa trên cấu hình.
2. **Máy khách NFS**: Máy khách sử dụng giao thức NFS để yêu cầu truy cập vào thư mục được chia sẻ từ máy chủ. Khi được cấp quyền, máy khách có thể đọc, ghi hoặc thực hiện các thao tác file khác như thể chúng nằm trên máy của mình.

**Ví dụ minh họa**: Giả sử bạn có một mạng với máy chủ NFS có địa chỉ IP là 192.168.1.100 và bạn muốn chia sẻ thư mục /var/nfs\_share:

1. Trên **máy chủ NFS**, bạn cấu hình thư mục để chia sẻ:
2. sudo mkdir /var/nfs\_share
3. sudo chown nobody:nogroup /var/nfs\_share
4. echo "/var/nfs\_share \*(rw,sync,no\_subtree\_check)" | sudo tee /etc/exports
5. sudo exportfs -a
6. Trên **máy khách NFS**, bạn mount thư mục từ máy chủ:
7. sudo mount 192.168.1.100:/var/nfs\_share /mnt

Sau khi thực hiện các bước trên, máy khách có thể truy cập vào /mnt và sẽ thấy nội dung của /var/nfs\_share từ máy chủ NFS như thể nó là một phần của hệ thống file cục bộ.

**Lợi ích của NFS**:

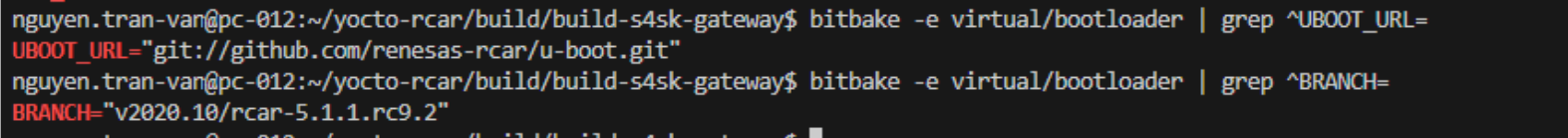
* **Dễ dàng chia sẻ dữ liệu**: NFS cho phép nhiều người dùng truy cập vào cùng một tập dữ liệu mà không cần phải sao chép hoặc di chuyển nó.
* **Tiết kiệm không gian lưu trữ**: Không cần phải lưu trữ các bản sao của cùng một file trên nhiều máy.
* **Hiệu quả trong quản lý**: Cập nhật và quản lý file từ một vị trí trung tâm.
* **Truy cập từ xa**: Truy cập file từ bất kỳ máy nào trong mạng mà không cần phải có file đó trên máy cục bộ.

[NFS rất hữu ích trong các môi trường mạng doanh nghiệp và giáo dục, nơi cần chia sẻ tài nguyên và dữ liệu giữa nhiều người dùng và hệ thống2](https://interdata.vn/blog/nfs-la-gi/).

**Full process từ burn u-boot tới load filesystem vào s4sk  
Build U-boot (thực hiện ở máy build để có đủ tool):**

Dựa vào các step trên để build u-boot với branch sau ([https://github.com/renesas-rcar/u-boot/blob/v2020.10/rcar-5.1.1.rc9.2/configs/**r8a779f0\_s4sk\_defconfig**](https://github.com/renesas-rcar/u-boot/blob/v2020.10/rcar-5.1.1.rc9.2/configs/r8a779f0_s4sk_defconfig) - file config nếu muốn mod lại u-boot):

git clone [https://github.com/renesas-rcar/u-boot.git -b v2020.10/rcar-5.1.1.rc9.2](https://github.com/renesas-rcar/u-boot.git -b%20v2020.10/rcar-5.1.1.rc9.2)

(làm sao để biết branch nào? Kiểm tra trong yocto build (yocto build sẽ nói ở phần sau) như sau

)

**Config local.conf file dựa trên poky version (poky là từ yocto project, đọc sau) (chỉ config khi sử dụng yocto để build u-boot)**

To configure our *conf/local.conf* file to work from the cloned source, modify it as follows:

INHERIT += "externalsrc"

EXTERNALSRC:pn-myrecipe = "/path/to/my/source/tree"

EXTERNALSRC:BUILD\_pn-myrecipe = "/path/to/my/source/tree"

Please refer [**externalsrc.bbclass**](https://git.yoctoproject.org/poky/plain/meta/classes/externalsrc.bbclass) file in Yocto source tree to get more information.

Note that each Poky version has different syntax (In this case Poky version 4)

Apply to S4SK board: (Poky version 3)

**local.conf file:**

INHERIT += "externalsrc"

EXTERNALSRC\_pn-u-boot = "/home/nguyen.tran-van/yocto-rcar/build/ext-src/u-boot"

EXTERNALSRC\_BUILD\_pn-u-boot = "/home/nguyen.tran-van/yocto-rcar/build/ext-src/uboot-build"

**How to find out Yocto version?**

Check out this file it gives you full details about Yocto version

vim $POKY-DIR/meta-poky/conf/distro/poky.conf

You will get info like:

DISTRO = "poky"

DISTRO\_NAME = "Poky (Yocto Project Reference Distro)"

DISTRO\_VERSION = "2.7.2"

DISTRO\_CODENAME = "warrior"

SDK\_VENDOR = "-pokysdk"

SDK\_VERSION = "${@d.getVar('DISTRO\_VERSION').replace('snapshot-${DATE}', 'snapshot')}"

.....

.....

Now you will know the version you are actually using.

**To find the kernel version you are using:**

bitbake -e virtual/kernel | grep "^PV"

Sau khi config xong thì make bằng cross-compiler (khi compile độc lập với yocto) ở trên hoặc xài lệnh: bitbake virtual/bootloader (chỉ dùng khi đang sử dụng yocto để build)

Khi make độc lập với yocto thì sẽ ra output là file **u-boot-elf.srec,** sửa lại thành **u-boot-elf-s4sk.srec** để burn ở bước sau:

cp u-boot-elf.srec u-boot-elf-s4sk.srec

**Nạp u-boot bằng tool unofficial "Renesas BSP ROM Writer" (chuyển file qua máy host dùng lệnh scp)**

Ví dụ về dùng scp để chuyển file : scp u-boot-elf-s4sk.srec <**username**>@192.168.2.[23:/path/to/ICUMX\_Loader\_and\_Flashwriter\_Package\_for\_R-Car\_S4\_Starter\_Kit\_SDKv3.16.0](http://0.0.0.23/path/to/ICUMX_Loader_and_Flashwriter_Package_for_R-Car_S4_Starter_Kit_SDKv3.16.0)

Tải tool , folder của tool lúc này sẽ là ${renesas-bsp-rom-writer} : git clone <https://github.com/morimoto/renesas-bsp-rom-writer>

Download ICUMX loader (giải nén hết vào cùng thư mục) and copy the built binaries (u-boot-elf-s4sk.srec, tee-s4sk.srec and bl31-s4sk.srec) có được ở trên phần build u-boot

[Link](https://www.renesas.com/us/en/icumx-loader-and-flash-writer-package-r-car-s4-starter-kit) tải ICUMX

Ta giải nén hết ICUMX loader và đem hết 3 file của u-boot vào thư mục đó, sau đó đảm bảo ta đứng ở thư mục chứa toàn bộ file này:

cd ${PATH}/ICUMX\_Loader\_and\_Flashwriter\_Package\_for\_R-Car\_S4\_Starter\_Kit\_SDKv3.16.xx/

Và từ đó trỏ vào file sdk\_writer trong tool:

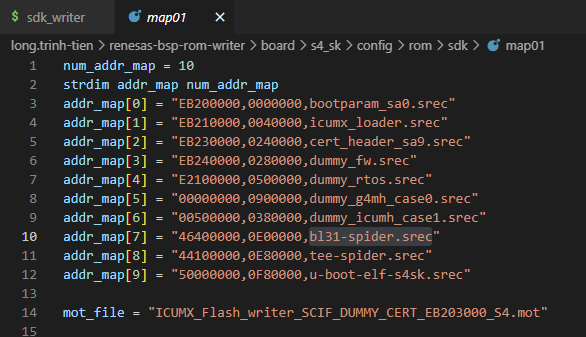
${renesas-bsp-rom-writer}/board/s4\_sk/linux/sdk\_writer



A screenshot of a computer program

Description automatically generated

Lưu ý ở bước này tool gốc bị 1 lỗi nhỏ, sửa lại như sau:

Ở tool rom-writer tại /home/long.trinh-tien/renesas-bsp-rom-writer/board/s4\_sk/config/rom/sdk/map01

Sửa spider thành s4sk tại addr\_map[7] và addr\_map[8]

A black background with white text

Description automatically generated

Ngay sau khi ấn y tại Power OFF thì bật terminal khác lên chạy cmd sau để vào burn mode (nhớ kêu mọi người thoát picocom, minicom hết nếu ko sẽ bị timeout hoặc crash):

A computer screen shot of a black screen

Description automatically generatedA screenshot of a computer program

Description automatically generatedcpld\_control\_v1.0 -w S4SK [ftdi://ftdi:2232:276697/2](ftdi://ftdi:2232:276697) 0x0008 0x00000080804922BF 0x0024 0x01

Sau đó chuyển qua terminal khác chạy cmd vào boot mode:

cpld\_control\_v1.0 -w S4SK [ftdi://ftdi:2232:276697/2](ftdi://ftdi:2232:276697) 0x0008 0x00000080804922A9 0x0024 0x01

More information:

Mình chia sẻ thêm một số thông tin về burn boot

**Manualy workflow:**  
Power off -> Gạt SW để chọn burn mode -> Turn on board -> Board đã vào burn mode và tiến hành burn -> Burn xong thì Power off -> Gạt SW để chọn boot mode và boot board

**Tooling workflow:**  
Dùng cpld\_tool để chuyển sang burn mode -> Board đã vào burn mode và tiến hành burn -> Burn xong thì dùng cpld\_tool để chuyển sang boot mode và boot board

Ở command:  
$ cpld\_control\_v1.0 -w S4SK ftdi://ftdi:2232:276697/2 0x0008 0x00000080804922BF 0x0024 0x01  
Command này sẽ làm 2 việc đó là chuyển board sang burn mode và reset board

$ cpld\_control\_v1.0 -w S4SK ftdi://ftdi:2232:276697/2 0x0008 0x00000080804922A9 0x0024 0x01  
Command này sẽ làm 2 việc đó là chuyển board sang boot mode và reset board

Nên việc power off hay power on ở tooling workflow có thể skip

Tool cpld này sẽ giúp chúng ta đổi mode mà không phải vào lab để gạt DIP SW trên board

**Load image**

Trên máy 2.23 mình note lại một số đường dẫn:

* rootfs ở đường dẫn:  /data/nfs/s4sk/
  + -> tìm file rcar-image-gateway-s4sk-20240502032430.rootfs.tar.bz2 (hoặc lấy file mà ta đã build trong yocto)
  + Tạo thư mục long.trinh-tien tại s4sk/ và chạy command để bung rootfs : tar -xjf rcar-image-gateway-s4sk-20240502032430.rootfs.tar.bz2
* Image + dtb ở đường dẫn:  /data/tftpboot/s4sk/01-2e-09-0a-00-c3-95/ (hoặc lấy file đã build trong yocto)
  + Copy trong default ra để tạo 1 thư mục riêng long.trinh-tien và copy vào đó (optional)
* Pxe boot (chưa rõ cách sử dụng) mọi người edit file 01-2e-09-0a-00-c3-95 ở đường dẫn:  /data/tftpboot/pxelinux.cfg/

Sau khi setup xong quay lại u-boot set các environment sau:

setenv serverip 192.168.2.23

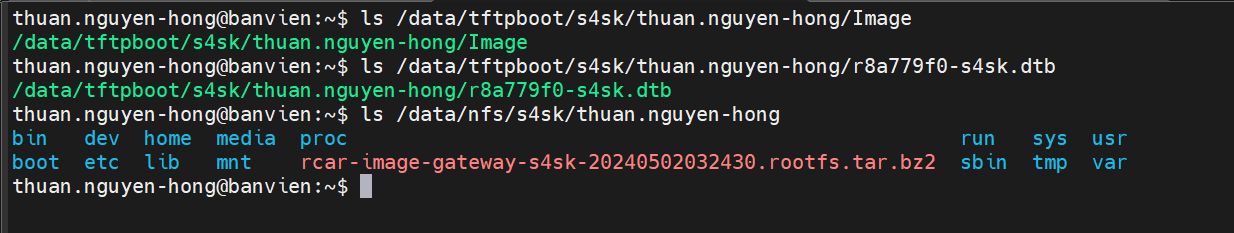
setenv ipaddr 192.168.2.95

setenv bootargs "rw root=/dev/nfs nfsroot=${serverip}:**/data/nfs/s4sk/long.trinh-tien**,nfsvers=4 ip=${ipaddr}:::::tsn0 ignore\_loglevel cma=560M"

setenv bootcmd 'tftpboot ${loadaddr} **s4sk/long.trinh-tien/Image**; tftpboot ${fdtaddr} **s4sk/long.trinh-tien/r8a779f0-s4sk.dtb**;booti ${loadaddr} - ${fdtaddr}'

saveenv

reset



giải thích các env trong U-boot:

+ipaddr : IP của board S4SK trong U-boot

+serverip: IP của server TFTP

+bootargs: dùng để boot board khi vào kernel

+bootcmd: sẽ được U-boot chạy khi boot board

Sau đó image của ta sẽ tự động được load

**Chi tiết thêm về việc setenv cho u-boot:**

More information:

TFTP boot, hay Trivial File Transfer Protocol boot, là một phương pháp khởi động hệ thống từ mạng mà không cần lưu trữ root filesystem (rootfs) trên thiết bị cục bộ như ổ cứng hoặc thẻ nhớ. [Thay vào đó, rootfs được lưu trữ trên một máy chủ mạng và được tải xuống thông qua mạng vào bộ nhớ RAM của thiết bị khi khởi động1](https://stackoverflow.com/questions/47100033/is-it-possible-tftp-in-u-boot-to-load-root-filesystem-to-sd-card-i-dont-want)[2](https://imxdev.gitlab.io/tutorial/How_to_boot_Kernel_and_File_System_from_the_network/).

Trong quá trình khởi động, bootloader như U-Boot sẽ sử dụng TFTP để tải kernel và các file cần thiết khác từ máy chủ mạng. [Sau đó, hệ thống có thể được cấu hình để mount rootfs từ một máy chủ NFS (Network File System), cho phép hệ thống hoạt động như thể rootfs đang được lưu trữ trên thiết bị cục bộ2](https://imxdev.gitlab.io/tutorial/How_to_boot_Kernel_and_File_System_from_the_network/).

[Điều này rất hữu ích trong quá trình phát triển phần mềm, vì nó cho phép các nhà phát triển thử nghiệm các thay đổi mà không cần phải ghi lại rootfs lên thiết bị lưu trữ cục bộ sau mỗi lần thay đổi1](https://stackoverflow.com/questions/47100033/is-it-possible-tftp-in-u-boot-to-load-root-filesystem-to-sd-card-i-dont-want). Tuy nhiên, để sử dụng rootfs từ máy chủ NFS, thiết bị cần duy trì kết nối mạng trong suốt quá trình hoạt động.

**Build yocto image:**

<https://elinux.org/R-Car/Boards/S4SK/Yocto-Linux/SDK-v3.16.2>

Để thêm package trong local.conf

[12 Variables Glossary — The Yocto Project ® 5.0.1 documentation](https://docs.yoctoproject.org/ref-manual/variables.html#term-IMAGE_INSTALL)

Thêm IMAGE\_INSTALL = “bc” (cài best calculator)

Rồi các package sau thì append vào( syntax thì đọc doc)

Kiểm tra các package có thể cài vào : [Useful bitbake commands - NXP Community](https://community.nxp.com/t5/i-MX-Processors-Knowledge-Base/Useful-bitbake-commands/ta-p/1128559)