Chương 1: Giới thiệu và Chuẩn bị

1.1. Giới thiệu

Tài liệu này trình bày chi tiết toàn bộ lộ trình kỹ thuật, bắt đầu từ việc sử dụng một hệ điều hành tiêu chuẩn có sẵn cho đến việc tự tay xây dựng, tùy chỉnh và nâng cấp một hệ điều hành Linux hoàn chỉnh. Toàn bộ quy trình được thực hiện trên một bo mạch nhúng cụ thể là Raspberry Pi 2 Model B. Mục tiêu là nắm vững các kỹ năng nền tảng trong việc phát triển hệ thống Linux nhúng.

1.2. Môi trường Thực nghiệm

Để thực hiện lộ trình, các thành phần phần cứng và phần mềm sau đã được sử dụng:

Phần cứng:

* Bo mạch: Raspberry Pi 2 Model B
* Nguồn: MicroUSB 5V - 1A
* Giao tiếp gỡ lỗi: Adapter USB to TTL CP2102
* Thiết bị lưu trữ: Thẻ nhớ SDCard 32GB và USB đọc thẻ nhớ

Phần mềm:

* Hệ điều hành máy chủ: Máy ảo Ubuntu 24.04
* Hệ thống Build: Yocto Project, phiên bản "Kirkstone"
* Công cụ ghi thẻ nhớ: Raspberry Pi Imager
* Chương trình Terminal: Tera Term

Chương 2: Giai đoạn I – Nền tảng và Xây dựng Tự động

2.1. Khởi động lần đầu với Raspberry Pi OS

Mục tiêu:

* Thực hiện kiểm tra và xác nhận bo mạch Raspberry Pi 2 hoạt động tốt.
* Cài đặt thành công một hệ điều hành Linux tiêu chuẩn lên thẻ nhớ.
* Xác định và nhận diện các thành phần cốt lõi của hệ điều hành (Image Kernel và DTB) ngay trên hệ thống đang chạy.

Quy trình:

1. **Chuẩn bị Thẻ nhớ:**
   * Phần mềm Raspberry Pi Imager được sử dụng để ghi hệ điều hành.
   * Phiên bản Raspberry Pi OS (Legacy, 32-bit) Lite được lựa chọn để đảm bảo tương thích với kiến trúc 32-bit của Pi 2.
   * Trong phần cài đặt nâng cao, Username và Password đã được thiết lập để phục vụ cho việc đăng nhập.
2. **Thiết lập Vật lý và Kết nối Hệ thống:**
   * Thẻ nhớ đã ghi HĐH được cắm vào bo mạch.
   * Kết nối vật lý được thiết lập giữa adapter USB to TTL và các chân GPIO UART của bo mạch.
   * Phần mềm Tera Term được mở trên máy tính, kết nối được thiết lập đến cổng Serial tương ứng với adapter. Tốc độ (Speed) được cấu hình thành 115200 trong mục Setup/Serial port.
   * Cấp nguồn MicroUSB để khởi động bo mạch.
   * Cửa sổ Tera Term hiển thị các dòng log khởi động, và việc xác thực đăng nhập được thực hiện thành công bằng mật khẩu đã thiết lập.

Kết quả:

Sau khi đăng nhập thành công, các thành phần cốt lõi của hệ điều hành đã được kiểm tra:

1. Di chuyển đến thư mục boot bằng lệnh: `cd /boot`
2. Liệt kê toàn bộ nội dung thư mục bằng lệnh: `ls -l`
3. Xác nhận thành công sự tồn tại của 2 file quan trọng:
   * `kernel7.img`: Được xác định là file Kernel của hệ thống. File này được biên dịch riêng để hoạt động trên kiến trúc ARMv7 của CPU trên Raspberry Pi 2.
   * `bcm2709-rpi-2-b.dtb`: Được xác định là file Device Tree Blob (DTB) của hệ thống. File này chứa "bản đồ phần cứng" dành riêng cho model bo mạch Raspberry Pi 2 Model B, vốn được trang bị chip Broadcom BCM2709.
4. Kết thúc phiên làm việc, hệ thống được tắt một cách an toàn bằng lệnh `sudo poweroff`.

2.2. Xây dựng HĐH Tối giản bằng Yocto

Mục tiêu:

* Thiết lập một môi trường build hoàn chỉnh sử dụng Yocto Project (phiên bản Kirkstone) trên hệ điều hành Ubuntu.
* Tự xây dựng một hệ điều hành Linux tối giản (`core-image-minimal`) từ mã nguồn, được tùy chỉnh riêng cho bo mạch Raspberry Pi 2.
* Triển khai thành công image vừa build lên thẻ nhớ và khởi động bo mạch.

Quy trình:

1. Thiết lập môi trường Build trên Ubuntu:

* Cập nhật danh sách gói và cài đặt các công cụ phụ thuộc cần thiết cho Yocto bằng các lệnh sau:

bash----------

```
sudo apt update
sudo apt install gawk wget git diffstat unzip texinfo gcc build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 libsdl1.2-dev python3-subunit mesa-common-dev zstd liblz4-tool file locales
sudo locale-gen en_US.UTF-8
```

---

2. Tải mã nguồn Yocto và các Layer:

* Một thư mục làm việc `~/yocto` đã được tạo.
* Các kho mã nguồn cần thiết cho phiên bản "Kirkstone" được tải về bằng Git:

bash----------

```
# Tải Poky (lõi của Yocto)
git clone -b kirkstone git://git.yoctoproject.org/poky.git

# Tải meta-openembedded
git clone -b kirkstone git://git.openembedded.org/meta-openembedded

# Tải layer cho Raspberry Pi
git clone -b kirkstone https://github.com/agherzan/meta-raspberrypi.git
```

---

3. Khởi tạo và Cấu hình Môi trường Build:

* Môi trường build được khởi tạo bằng cách di chuyển vào thư mục `~/yocto/poky` và chạy lệnh `source oe-init-build-env`.
* File `conf/bblayers.conf` được cấu hình để Yocto nhận diện các layer đã tải. Nội dung file như sau:

```
makefile----------------------------------------------
POKY_BBLAYERS_CONF_VERSION = "2"
BBPATH = "${TOPDIR}"
BBFILES ?= ""
BBLAYERS ?= " \
  /home/danhdelrey/yocto/poky/meta \
  /home/danhdelrey/yocto/poky/meta-poky \
  /home/danhdelrey/yocto/poky/meta-yocto-bsp \
  /home/danhdelrey/yocto/meta-openembedded/meta-oe \
  /home/danhdelrey/yocto/meta-openembedded/meta-python \
  /home/danhdelrey/yocto/meta-raspberrypi \
  "
----------------------------------------------
```

* File `conf/local.conf` được chỉnh sửa để chỉ định máy đích và thêm các tính năng cơ bản. Các dòng sau đã được thêm vào cuối file:

```
makefile----------------------------------------------
# === CẤU HÌNH CHO RASPBERRY PI 2 MODEL B ===
MACHINE ?= "raspberrypi2"
ENABLE_UART = "1"
EXTRA_IMAGE_FEATURES += "ssh-server-openssh"
----------------------------------------------
```

4. Thực thi Build:

* Do Ubuntu 24.04 có các chính sách bảo mật mới, lệnh sau đã được chạy để cho phép Yocto tạo môi trường cách ly (sandbox):

```
bash--------------------------------------------------
sudo sysctl -w kernel.unprivileged_userns_clone=1
--------------------------------------------------
```

* Quá trình build được bắt đầu bằng lệnh: `bitbake core-image-minimal`.
* Thông tin cấu hình build đã được xác nhận qua log, cho thấy các thông số như `MACHINE = "raspberrypi2"` và `NATIVELSBSTRING = "ubuntu-24.04"` là chính xác.

5. Chuẩn bị Image để triển khai:

* Sau khi build xong, các file kết quả nằm tại `~/yocto/poky/build/tmp/deploy/images/raspberrypi2`.
* File image gốc được nén dưới dạng `.wic.bz2`. Do `bunzip2` có cơ chế an toàn ngăn giải nén file đang có symlink trỏ tới, một bản sao đã được tạo trước:

```
bash--------------------------------------------------
cp core-image-minimal-raspberrypi2-*.rootfs.wic.bz2 my-rpi2-image.wic.bz2
--------------------------------------------------
```

* Bản sao sau đó được giải nén để có được file image thô:

```
bash--------------------------------------------------
bunzip2 my-rpi2-image.wic.bz2
--------------------------------------------------
```

* Kết quả thu được là file `my-rpi2-image.wic` với dung lượng khoảng 90MB.

6. Ghi Image ra thẻ nhớ và chạy trên board:

* Phần mềm Raspberry Pi Imager được sử dụng để ghi file `my-rpi2-image.wic` tùy chỉnh vào thẻ nhớ.
* Bo mạch đã được khởi động thành công với hệ điều hành mới.

Kết quả:

* Xác nhận phiên bản Kernel:

```
---------------------------------------------
root@raspberrypi2:~# uname -a
Linux raspberrypi2 5.15.92-v7 #1 SMP Wed Feb 8 16:47:50 UTC 2023 armv7l GNU/Linux
---------------------------------------------
```

* Xác nhận dung lượng hệ thống tối giản:

```
---------------------------------------------
root@raspberrypi2:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root      26.4M 10.9M 13.4M  45% /
/dev/mmcblk0p1 49.8M 28.5M 21.3M  57% /boot
---------------------------------------------
```

2.3. Nâng cao: Build Image `core-image-base` và Triển khai Thủ công

Mục tiêu:

* Nắm vững quy trình phân vùng và định dạng một thiết bị lưu trữ trên Linux.
* Hiểu rõ cấu trúc phân vùng tối thiểu cần có để một bo mạch Raspberry Pi có thể khởi động.
* Học cách triển khai Kernel, Device Tree và Rootfs một cách thủ công từ các thành phần riêng lẻ.
* Sử dụng cổng Serial để gỡ lỗi khi hệ thống không khởi động được.

Quy trình:

Bước 1: Kích hoạt Môi trường và Thực thi Lệnh Build

1. Môi trường build Yocto được kích hoạt lại trên máy ảo Ubuntu.
2. Lệnh `sudo sysctl -w kernel.unprivileged_userns_clone=1` được chạy để cho phép tạo User Namespace.
3. File `conf/local.conf` được kiểm tra và bổ sung các cấu hình sau để thêm trình quản lý gói `opkg`:

   makefile----------
   # Thêm SSH server và trình quản lý gói opkg
   EXTRA_IMAGE_FEATURES += "ssh-server-openssh package-management"
   # Chỉ định Yocto build các gói theo định dạng .ipk
   PACKAGE_CLASSES = "package_ipk"
   ----

4. Lệnh build được thực thi để tạo image `core-image-base`:

   bash----------
   bitbake core-image-base

Bước 2: Thu hoạch Thành phẩm (Linh kiện rời)

Sau khi build xong, các thành phần riêng lẻ cần thiết cho việc triển khai thủ công được xác định tại `~/yocto/poky/build/tmp/deploy/images/raspberrypi2/`:

1. Rootfs (Ổ cứng chứa HĐH): File `core-image-base-raspberrypi2.tar.bz2`.
2. Kernel (Bộ não): File `zImage`.
3. Device Tree (Bản đồ): File `bcm2709-rpi-2-b.dtb`.
4. Firmware & Bootloader (Bộ mồi khởi động): Các file `bootcode.bin`, `start.elf`, và `fixup.dat` (nằm trong thư mục con `bootfiles`).

Bước 6 - 8: Chuẩn bị Thẻ nhớ (Phân vùng và Định dạng)*

1. Thẻ nhớ được kết nối vào máy ảo Ubuntu và được xác định tên thiết bị (ví dụ: `/dev/sdb`) bằng lệnh `lsblk`.
2. Bảng phân vùng cũ trên thẻ nhớ được xóa sạch để chuẩn bị:

   bash----------
   sudo umount /dev/sdb*
   sudo wipefs --all /dev/sdb
   ----

3. Công cụ `fdisk` được sử dụng để tạo 2 phân vùng mới:
   * Phân vùng 1 (BOOT): Được tạo với kích thước `+100M` và kiểu `c` (W95 FAT32 LBA).
   * Phân vùng 2 (ROOTFS): Được tạo để sử dụng toàn bộ dung lượng còn lại của thẻ.
   * Các thay đổi được lưu lại bằng lệnh `w`.
4. Hai phân vùng thô vừa tạo (`/dev/sdb1` và `/dev/sdb2`) được định dạng:
   * Phân vùng BOOT được format thành FAT32:

     bash----------
     sudo mkfs.vfat -n BOOT /dev/sdb1
     ----

* Phân vùng ROOTFS được format thành ext4:

bash----------
sudo mkfs.ext4 -L rootfs /dev/sdb2

Bước 3: Chép File vào các Phân vùng (Lắp ráp)

1. Hai phân vùng được mount vào hệ thống file của Ubuntu:

bash----------
mkdir ~/boot_mount ~/rootfs_mount
sudo mount /dev/sdb1 ~/boot_mount
sudo mount /dev/sdb2 ~/rootfs_mount
---

2. Các file Bootloader, Firmware, Kernel, và DTB được chép vào phân vùng BOOT. File Kernel `zImage` được đổi tên thành `kernel7.img` theo yêu cầu của firmware Raspberry Pi:

bash----------
Di chuyển đến thư mục chứa linh kiện
cd ~/yocto/poky/build/tmp/deploy/images/raspberrypi2
# Chép bootloader và firmware
sudo cp bootfiles/* ~/boot_mount/
# Chép Kernel và đổi tên
sudo cp zImage ~/boot_mount/kernel7.img
# Chép Device Tree
sudo cp bcm2709-rpi-2-b.dtb ~/boot_mount/
---

3. File `cmdline.txt` được tạo và thêm vào phân vùng BOOT với nội dung sau để chỉ dẫn cho Kernel:

---
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 root=/dev/mmcblk0p2 rootwait
---

4. Rootfs được giải nén trực tiếp vào phân vùng ROOTFS:

bash----------
sudo tar -xvpf core-image-base-raspberrypi2.tar.bz2 -C ~/rootfs_mount/
---

5. Các phân vùng được unmount an toàn và thẻ nhớ được gắn vào bo mạch để khởi động.

Kết quả:

Bo mạch đã khởi động thành công với hệ điều hành được triển khai thủ công. Quá trình khởi động được theo dõi và việc đăng nhập được thực hiện qua cổng Serial, xác nhận hệ thống hoạt động ổn định. Kỹ năng phân vùng và triển khai thủ công đã được nắm vững.

Chương 3: Giai đoạn II – "Phẫu thuật" Kernel

3.1. Tìm công thức Kernel

Mục tiêu:

* Xác định phiên bản: Tìm ra chính xác mã commit của mã nguồn Kernel mà Yocto đã sử dụng.
* Xác định nguồn gốc: Tìm ra địa chỉ kho Git mà Yocto đã tải mã nguồn.
* Thu thập thông tin: Lấy đủ thông tin cần thiết để có thể tải về một bản sao y hệt của mã nguồn đó.

Quy trình:

1. **Quay trở lại Môi trường Build:**
   * Môi trường Yocto được kích hoạt lại trên Terminal.
2. **Tìm file Recipe của Kernel:**
   * Lệnh `find` được sử dụng để tìm kiếm file công thức trong toàn bộ thư mục `yocto`:
     bash----------
     cd ~/yocto
     find . -name "linux-raspberrypi*.bb"
     ----
   * Kết quả cho thấy công thức cho phiên bản Kernel `5.15` nằm tại: `./meta-raspberrypi/recipes-kernel/linux/linux-raspberrypi_5.15.bb`.
3. **Đọc và Phân tích Recipe:**
   * File công thức trên được mở bằng trình soạn thảo văn bản.
   * Các biến quan trọng sau đã được phân tích và ghi nhận:
     * Biến `SRC_URI`: Cho biết nguồn gốc mã nguồn.
       * Kho Git: `https://github.com/raspberrypi/linux.git`
       * Tên Branch: `rpi-5.15.y`
     * Biến `SRCREV`: Chỉ định mã commit chính xác.
       * Mã Commit: Dãy 40 ký tự hexa được ghi lại từ biến `SRCREV_machine`.

Kết quả:

Đã thu thập thành công 3 thông tin quan trọng cần thiết để tái tạo lại môi trường biên dịch Kernel một cách chính xác.

3.2. Biên dịch chéo Kernel 5.15

Mục tiêu:

* Tải về bản sao mã nguồn Kernel chính xác mà Yocto đã sử dụng.
* Thiết lập một môi trường biên dịch chéo (cross-compilation) ổn định và tương thích trên máy chủ Ubuntu.
* Thực hiện biên dịch thành công mã nguồn Kernel để tạo ra các file nhị phân cần thiết (`zImage`, `modules`, `dtbs`) cho Raspberry Pi 2.

Quy trình:

1. Cài đặt các Công cụ và Thư viện Phụ thuộc:

* Một bộ công cụ phát triển đầy đủ, bao gồm các thư viện cần thiết cho việc biên dịch Kernel, đã được cài đặt:

```
bash----------------------------------------------
sudo apt update
sudo apt install build-essential libncurses-dev bison flex libssl-dev libelf-dev libgmp-dev libmpc-dev libmpfr-dev
----------------------------------------------
```

2. Cài đặt Bộ Công cụ Biên dịch chéo:

* Do phát hiện sự không tương thích giữa SDK của Yocto Kirkstone và môi trường Ubuntu 24.04, một toolchain tiêu chuẩn đã được cài đặt từ kho phần mềm của Ubuntu để đảm bảo tính tương thích:

```
bash----------------------------------------------
sudo apt install crossbuild-essential-armhf
----------------------------------------------
```

3. Tải Mã nguồn Kernel:

* Một thư mục làm việc mới (`~/kernel_dev`) được tạo.
* Lệnh `git clone` được sử dụng để tải về mã nguồn Kernel từ kho và branch đã xác định ở trên:

```
bash----------------------------------------------
git clone --depth 1 --branch rpi-5.15.y https://github.com/raspberrypi/linux.git
----------------------------------------------
```

4. Thực hiện Biên dịch:

* Một Terminal mới được mở để đảm bảo môi trường sạch.
* Các biến môi trường được thiết lập để chỉ định kiến trúc đích và bộ công cụ biên dịch sẽ sử dụng:

```
bash----------------------------------------------
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
----------------------------------------------
```

* Mã nguồn được dọn dẹp (`make mrproper`) và cấu hình mặc định cho Raspberry Pi 2 được nạp (`make bcm2709_defconfig`).
* Quá trình biên dịch được bắt đầu bằng lệnh:

```
bash----------------------------------------------
make -j$(nproc) zImage modules dtbs
----------------------------------------------
```

Kết quả:
Quá trình biên dịch đã hoàn thành thành công. Đã kiểm tra và xác nhận sự tồn tại của các sản phẩm đầu ra quan trọng:
* File Kernel nén:
  `danhdelrey@danhdelrey-X79G:~/kernel_dev/linux$ ls -l arch/arm/boot/zImage`
  `-rwxrwxr-x 1 danhdelrey danhdelrey 6652816 Sep 10 20:20 arch/arm/boot/zImage`
* File Device Tree Blob:
  `danhdelrey@danhdelrey-X79G:~/kernel_dev/linux$ ls -l arch/arm/boot/dts/bcm2709-rpi-2-b.dtb`
  `-rw-rw-r-- 1 danhdelrey danhdelrey 29305 Sep 10 20:11 arch/arm/boot/dts/bcm2709-rpi-2-b.dtb`

3.3. "Cấy ghép" Kernel và Thử nghiệm

Mục tiêu:
* Cập nhật các file Kernel, Device Tree và các modules trên thẻ nhớ bằng các file vừa biên dịch.
* Kiểm tra: Khởi động thành công Raspberry Pi với Kernel mới.
* Đảm bảo hệ thống vẫn hoạt động ổn định và xác nhận phiên bản Kernel đã được thay đổi.

Quy trình:
Quá trình này được thực hiện như một "ca phẫu thuật có chọn lọc", chỉ thay thế các thành phần liên quan đến Kernel và giữ nguyên phân vùng `rootfs` đã ổn định.
1. **Chuẩn bị Thẻ nhớ và Mount các Phân vùng:**
   * Thẻ nhớ đã được triển khai `core-image-base` thủ công được sử dụng.
   * Sau khi kết nối thẻ nhớ vào máy ảo Ubuntu, cả hai phân vùng (`/dev/sdb1` và `/dev/sdb2`) được mount vào các thư mục `~/boot_mount` và `~/rootfs_mount`.
2. **"Cấy ghép" - Sao chép các File mới:**
   * Di chuyển vào thư mục mã nguồn Kernel (`~/kernel_dev/linux`).
   * Môi trường biên dịch được kích hoạt lại với các biến `ARCH` và `CROSS_COMPILE`.
   * Cài đặt các Modules: Các driver (`.ko`) đã biên dịch được cài đặt vào đúng vị trí trong `rootfs` bằng lệnh:

```
bash------------------------------------------------
sudo make modules_install INSTALL_MOD_PATH=~/rootfs_mount
------------------------------------------------
```

Lệnh này tạo ra một thư mục phiên bản mới trong `~/rootfs_mount/lib/modules/` và chép tất cả các file driver vào đó.
   * Sao chép Kernel và Device Tree: Các file `zImage` và `.dtb` mới được chép vào phân vùng BOOT, đồng thời các file cũ được sao lưu để đảm bảo an toàn. File `zImage` được đổi tên thành `kernel7.img`.

```
bash------------------------------------------------
# Sao lưu Kernel cũ
sudo mv ~/boot_mount/kernel7.img ~/boot_mount/kernel7-backup.img
# Chép Kernel mới và đổi tên
sudo cp arch/arm/boot/zImage ~/boot_mount/kernel7.img
# Sao lưu Device Tree cũ
sudo mv ~/boot_mount/bcm2709-rpi-2-b.dtb ~/boot_mount/bcm2709-rpi-2-b-backup.dtb
# Chép Device Tree mới
sudo cp arch/arm/boot/dts/bcm2709-rpi-2-b.dtb ~/boot_mount/
------------------------------------------------
```

3. Hoàn tất và Dọn dẹp:

* Lệnh `sync` được chạy để đảm bảo dữ liệu được ghi xuống thẻ nhớ.
* Các phân vùng được unmount an toàn.

Kết quả:

* Bo mạch đã khởi động thành công với Kernel mới.
* Sau khi đăng nhập qua cổng Serial, phiên bản Kernel mới đã được xác nhận bằng lệnh `uname -a`. Kết quả trả về một chuỗi định danh mới với dấu thời gian khớp với thời điểm biên dịch, chứng tỏ "ca cấy ghép" đã thành công mỹ mãn.

```
---------------------------------------------
root@raspberrypi2:~# uname -a
Linux raspberrypi2 5.15.92-v7+ #1 SMP Wed Sep 10 20:11:12 +07 2025 armv7l GNU/Linux
---------------------------------------------
```

Tổng kết các Kỹ thuật Trọng tâm

Kỹ thuật	Mục đích	Kết quả Thực tế
1. Sử dụng Hệ thống Build (Yocto)	Để tự động hóa việc xây dựng một hệ điều hành Linux tùy chỉnh hoàn chỉnh từ hàng ngàn gói mã nguồn.	Đã tự tay build thành công một image core-image-base nhỏ gọn nhưng đầy đủ tính năng (opkg, ssh), chứng tỏ khả năng tạo ra một rootfs theo yêu cầu.
2. Biên dịch chéo (Cross-Compilation)	Để tạo ra các chương trình (Kernel, driver) chạy được trên kiến trúc CPU của bo mạch đích (ARM) bằng cách sử dụng một máy tính mạnh hơn có kiến trúc khác (x86_64).	Đã cài đặt và sử dụng thành công toolchain arm-linux-gnueabihf- để biên dịch Kernel 6.1. Kết quả là các file zImage, .dtb và modules chạy được trên Pi 2.
3. Phân vùng & Định dạng Thủ công	Để hiểu rõ cấu trúc lưu trữ tối thiểu cần có để một hệ thống Linux có thể khởi động, không bị phụ thuộc vào các công cụ tự động như file .wic.	Đã sử dụng fdisk và mkfs để tạo thành công hai phân vùng (BOOT-FAT32, ROOTFS-ext4) trên thẻ nhớ. Đã triển khai thủ công các file và khởi động được hệ thống.
4. Chỉnh sửa Device Tree (DTS/DTB)	Để mô tả và điều khiển phần cứng cho Kernel một cách linh hoạt mà không cần phải sửa đổi mã nguồn của Kernel. Đây là cách hiện đại để cấu hình phần cứng.	Đã chỉnh sửa thành công file .dts để vô hiệu hóa đèn LED trạng thái (ACT LED), sau đó biên dịch lại và xác nhận thay đổi có hiệu lực trên bo mạch thật.
5. Phân tích Mã nguồn (Driver & Device Tree)	Để hiểu được mối liên kết logic giữa mô tả phần cứng (DTS) và đoạn code thực thi (Driver C), từ đó có khả năng gỡ lỗi và tùy chỉnh sâu hơn.	Đã lần theo "dấu vết" của thuộc tính compatible, từ đó tìm ra chính xác các file mã nguồn và hàm probe cho driver của SD Card và Serial Port.
6. Porting Kernel	Để nâng cấp thành phần cốt lõi của hệ điều hành lên một phiên bản mới hơn.	Đã nâng cấp thành công Kernel của Raspberry Pi 2 từ phiên bản 5.15 (của Yocto) lên phiên bản 6.1 (tự biên dịch), và khởi động thành công bo mạch.