임베디드 리눅스 프로그래밍 완전 정복

임베디드 리눅스

- 선택 이유
  - 멀티 태스킹/인터넷 프로토콜 스택/USB, WiFi, 블루투스 및 다양한 저장 장치를 비롯 멀티미디어 장치도 지원
  - ARM, MIPS, x86, PPC 등 다양한 SoC 칩 지원
  - 오픈소스이므로 필요에 따라 수정 및 삭제가 용이
  - 매우 활성화 되어있는 커뮤니티 활동에 따라 최신의 하드웨어, 프로토콜, 표준을 지원
  - 특정 벤더에 종속되지 않음.

임베디드 리눅스

- 선택시 고려 사항
  - 임베디드 하드웨어 사양이 리눅스 실행에 충분한지 검토
  - 리눅스 자체에 대한 기술과, 하드웨어와 리눅스 사이의 관계에 이해 필요
  - 실시간 시스템 구성
  - 코드에 대한 규제 및 승인이 필요한 경우, 상업적으로 사용가능한 배포판이나 다른 OS 선택

임베디드 리눅스 주요 단체

- 주요 오픈소스 프로젝트 커뮤니티
  : 리눅스 커널, U-Boot, BusyBox, BuildRoot, Yocto, GNU . . .
- CPU 아키텍트
  : ARM/Linaro(Cortex-M), 인텔(x86, x86_64), sifive((Risc-V), IBM(PowerPC)
- SoC 벤더
  : 브로드컴, 인텔, 마이크로칩, NXP, 퀄컴, TI 등
- 보드 벤더
  : BeagleBoard/BeagleBone, RaspberryPi 등
- 상업용 리눅스 공급 업체
  : 지멘스, 타임시스, 윈드리버 등 상업용 리눅스 배포판 지원

임베디드 리눅스 4가지 요소

- 툴 체인 : 타깃 장치를 위한 컴파일러 및 개발 도구
- 부트 로더 : 보드 초기화와 리눅스 커널을 로드
- 커널 : 시스템 자원 관리 및 하드웨어 제어
- 루트 파일시스템 : 커널 초기화 이후 실행되는 프로그램 및 라이브러리가 들어있는 시스템
- 애플리케이션

오픈 소스 라이선스

- 오픈소스 라이선스 가이드 참고

- GPL 라이선스 : Copyleft 라이선스
- BSD, MIT 라이선스 : Permissive 라이선스

임베디드 리눅스를 위한 하드웨어

- 커널이 지원하는 CPU 아키텍처
- 적절한 크기의 RAM : 최소 16MiB 이상
- 비휘발성 메모리(플래시 메모리) : 8MiB 정도면 간단한 웹캠 또는 라우터 장치 가능
- 디버그 포트 : UART 기반의 시리얼 포트 선호,
- 소프트웨어를 장치로 로딩할 방법 : 예전에는 JTAB 인터페이스 많이 사용, 최근에는 착탈식 매체(예, SD카드) 또는 시리얼 인터페이스를 이용

QEMU(Quick-Emulator)

https://www.qemu.org/docs/master/index.html

- 가상 머신 및 에뮬레이션 환경을 만들기 위해 Fabrice Bellard가 개발한 오픈 소스 소프트웨어
- x86 기반 PC에서 ARM 소프트웨어를 실행하는 등 다양한 하드웨어 플랫폼용으로 설계된 운영 체제 및 애플리케이션을 실행할 수 있게 해 줌.
- VirtualBox와 같은 하이퍼바이저의 기반 기술

- 패키지 종류

```
qemu-system-arm : ARM/ARM64
qemu-system-mips : MIPS
qemu-system-ppc : PowerPC
qemu-system-x86 : x84/x86_64
```

ARM 이키텍처를 위한 QEMU 설치

- 패키지 설치

```
$ sudo apt-get install qemu-system-arm bridge-utils
```

- Linux 실행 파일만 에뮬레이션하는 경우

```
$ sudo apt-get install qemu-user-static

# 소스코드 크로스 컴파일 (** 컴파일시 –static 옵션 필요)
# 실행
$ qemu-⟨architecture⟩–static 실행파일명 [실행인자, ….]
# 디버깅
$ gdb-multiarch 실행파일명
```

qemu 가상환경 구성1-1

- 우분투 클라우드 이미지 다운로드

```
https://cloud-images.ubuntu.com/minimal/releases/noble/release/
```

- EFI 펌웨어를 부팅할 때 필요한 NVRAM 변수를 저장하기 위한 VM별 플래시 볼륨 생성

```
$ truncate -s 64m varstore.img
```

- ARM UEFI 펌웨어 복사

```
$ truncate -s 64m efi.img
$ dd if=/usr/share/qemu-efi-aarch64/QEMU_EFI.fd  of=efi.img  conv=notrunc
```

qemu 가상환경 구성1-2

- 리눅스 클라우드 이미지의 root 계정의 비밀번호 설정

```
$ sudo apt install libguestfs-tools
$ sudo virt-customize -a ubuntu-24.04-minimal-cloudimg-arm64.img  --root-password
password:⟨pass⟩
```

- qemu를 이용한 ARM64 비트 ubuntu24.04-minimal 클라우드 이미지 기반 가상머신 구동

```
$ sudo qemu-system-aarch64  -m 2048 -cpu max -M virt -nographic \
 -drive if=pflash,format=raw,file=efi.img,readonly=on \
 -drive if=pflash,format=raw,file=varstore.img \
 -drive if=none,file=ubuntu-24.04-minimal-cloudimg-arm64.img,id=hd0 \
 -device virtio-blk-device,drive=hd0 \
 -netdev type=tap,id=net0 \
 -device virtio-net-device,netdev=net0
```

qemu 가상 환경 구성-2

- 64비트에서 QEMU를 사용하여 Raspberry Pi 3 에뮬레이션

https://farabimahmud.github.io/emulate-raspberry-pi3-in-qemu/

1. Launch Script: launch.sh
2. Kernel: kernel8.img
3. DTB: bcm2710-rpi-3-b-plus.dtb
4. Disk Image (disk.img): You can use download either the .xz format or the .img format directly -

   a. disk.xz Archive .xz format (~833MB) need to decompress using (`$ tar xvJf disk.xz`)

   b. disk.img Decompressed .img format (~4G)

Username: pi
Password: raspberry

for rpi4
https://www.iot-tech.dev/phpBB3/viewtopic.php?t=311&sid=b4051a72744b961adac21b69c2a3e6a9

2장 툴체인

- 패키지 준비

```
$ sudo apt-get install autoconf automake bison bzip2 cmake flex g++ \
  gawk gcc gettext git gperf help2man libncurses5-dev libstdc++6 libtool \
  libtool-bin make patch python3-dev rsync texinfo unzip wget xz-utils
```

- 툴체인 이란 컴파일러, 링커, 런타임 라이브러리를 포함하는 컴파일 도구의 집합이다.
- 리눅스용 툴체인

➢ GCC 기반의 GNU 프로젝트 : GPL 라이선스

➢ Clang 컴파일러 기반의 LLVM 프로젝트 : BSD 라이선스

https://www.kernel.org/doc/html/latest/kbuild/llvm.html 참조
https://clang.llvm.org/docs/CrossCompilation.html 참조

표준 GNU 툴체인

- binutils : 어셈블러와 링커를 포함한 바이너리 유틸리티
  (http://gnu.org/software/binutils)

- GNU 컴파일러 컬렉션 (GCC): C 언어를 비롯한 C++, Objective-C, Objective-C++, Java, Fortran, Ada, Go 를 위한 컴파일러. (http://gcc.gnu.org/)

- C 라이브러리 : POSIX 사양을 기반으로 하는 표준화된 애플리케이션 프로그램 인터페이스 (API)로, 애플리케이션을 위한 운영 체제 커널에 대한 주요 인터페이스

- 이외에 커널 헤더 필요(5장 루트파일시스템 만들기 참조)

툴체인 종류

- 네이티브 툴체인
  : 툴체인이 만들어 내는 프로그램과 같은 종류의 시스템, 또는 실제로 같은 시스템인 경우
- 크로스 툴체인
  : 툴체인이 타깃 시스템 과 다른 종류의 시스템에서 실행 되는 경우
- 툴체인은 CPU 아키텍처에 맞게 빌드
  - ARM, MIPS, x86_64등
  - 빅 엔디언 or 리틀 엔디언
  - 부동소수점 지원
  - ABI(함수 호출간 인자를 넘기는 호출 규칙)

툴체인 찾기

- 미리 빌드된 툴체인
  - SoC 또는 보드 벤더에서 대부분 제공
  - 서드파티 벤더 제공(멘토그래픽스, 타임시스, 몬타비스타 등)
  - 리눅스 배포판에서 제공하는 크로스툴체인 패키지
  - 통한 임베디드 빌드 도구에서 제공하는 SDK(예, Yocto)

직접 크로스툴체인 빌드하는 경우

- crosstool-NG 설치하여 크로스 툴체인

```
$ git clone https://github.com/crosstool-ng/crosstool-ng.git
$ cd crosstool-ng
$ ./bootstrap
$ ./configure –prefix=${PWD}
$ make
$ make  install

$ bin/ct-ng  list-samples
$ bin/ct-ng show-〈가장유사한sample config 명〉   # 변경해야할 사항 check
$ bin/ct-ng 〈가장유사한sample config 명〉
$ bin/ct-ng menuconfig                                  # 변경 적용
$ bin/ct-ng build                                       # x-tools/ 디렉토리에 툴체인 생성
```

3. 부트로더

- 부트로더란 시스템을 시작하고 운영체체 커널을 로딩하는 프로그램
- 부트로더와 커널 인터페이스 : 아키텍처별로 상이

  부트로더의 하드웨어 구성정보와 커널의 명령 줄 전달

- 커널이 시작되면 부트로더는 메모리로부터 삭제
- 부트로더의 부 기능

  부트 구성 업데이트
  새로운 부트 이이지 메모리로 로드
  진단 기능 등
  주로 시리얼 인터페이스 명령줄 형태로 제공

부트로더의 역할

- 타겟 시스템 초기화
  - 전원이 입력되면 타겟 시스템이 정상동작 할 수 있도록 하드웨어 및 소프트웨어 동작 환경을 설정
  - 불필요한 하드웨어의 동작 중지, 시스템 클럭 설정. 메모리 콘트롤러 설정 및 필요시 MMU 또는 MPU 설정
  - 프로그램 동작에 필요한 재배치(relocation), 스택 영역 설정 및 C에서 사용되는 변수 영역을 설정한 다음 함수 호출
  - 필요에 따라 IRQ와 같은 예외처리(Exception Handling) 처리 백터 및 핸들러 작성

- 타겟 시스템 동작 환경 설정
  - 부트 방법 , 부트 디바이스를 비롯한 네트워크를 이용한 부트를 지원하기 위한 네 트워크 설정, ip 주소 설정 등동작에 필요한 정보를 설정

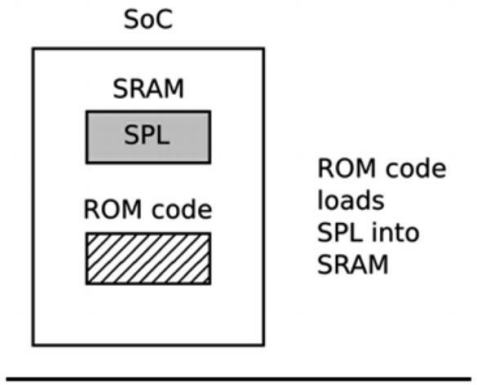
부트로더의 기능

- 시스템 운영체제 부팅
  - 일반적으로 임베디드 시스템의 운영체제는 플래시 메모리에 탑재되어 있고 부팅과정에서 주 메모리 (일반적으로 DRAM을)에 탑재하여 실행
  - 운영체제를 DRAM에 복사하고 제어권을 운영체제의 시작점으로 넘겨주는 기능 필요
  - 시리얼, 네트워크 또는 USB등을 이용하여 다운로드하는 방법 지원

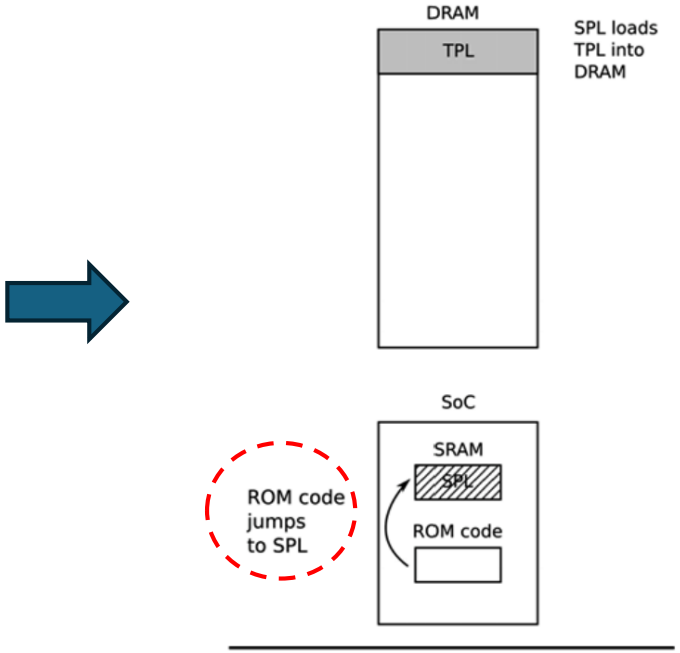
- 플래시 메모리 관리
  - 임베디드 시스템에서 가장 효율적인 보조 기억 장치
  - 부트로더 및 시스템 동작에 필요한 환경 변수저장
  - 시스템 운영체제 이미지 탑재

- 모니터기능
  - 시스템의동작 상태를 감시
  - POST(Power-On Self Test)
    하드웨어 정상 동작 여부 검사 및 메모리 검사 등

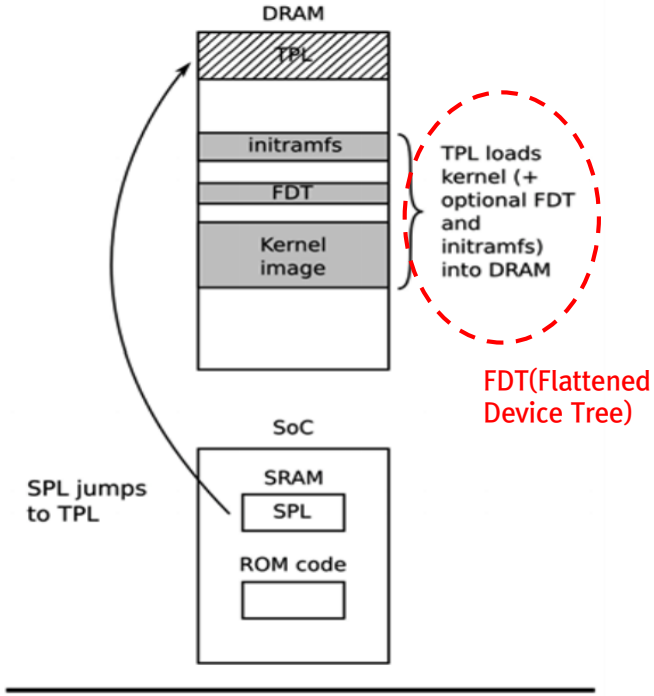
부트 순서



1단계 : ROM 코드



2단계 : SPL
(Secondary Program Loader)



3단계 : TPL
(Tertiary Program Loader)

부트 로더에서 커널로

- 부트로더가 커널로 제어를 넘길 때 전달해야 하는 정보

✓ 장치 트리가 지원되지 않는 아키텍처의 경우 SoC 식별을 위한 Machine Number

✓ 부트로더에서 감지된 하드웨어의 기본 사항(CPU 클럭 속도, 물리적인 RAM 크기 등)

✓ 커널 명령줄

✓ 장치 트리의 바이너리 위치와 크기

✓ 초기 램 디스크(램 파일시스템)의 위치와 크기

optional

장치 트리

- 컴퓨터 시스템의 하드웨어 요소를 정의하는 데이터 스트럭처
- 실행 코드가 아니라 정적인 데이터 임
- 일반적으로 부트 로더가 장치 트리를 로드하여 커널에 전달
- 장치 트리를 로드할 수 없는 부트로더의 경우 커널 이미지 자체에 포함 시킬 수도 있음.

←→ platform device 기반의 board 기술 방식(C coding)

〈등장 배경〉
- SoC 혹은 board 별로 독자적인 code 구현
- 같은 SoC에서 파생된 보드 간에 상호 연관성이 있음에도 불구하고, 이를 전혀 고려하지 않고, 별도로 구현
- 따라서, 코드의 복잡도 및 코드량이 늘어는 문제 발생함.
  : arch/arm/mach-*〈보드명〉*/board-*.c 파일이 매우 복잡하고 관리에 어려움.
  : ARM Linux 진영의 골칫거리로 대두 (Linus Torvalds의 지적)
- 보드 구성이 바뀌더라도 kernel code를 수정하지 않고, 동작할 수 있는 방식의 필요성 인식
- Device Tree는 기존에 다른 쪽(CPU)에서 사용하던 방식으로 ARM에도 채용하게 됨.
  : 새로 나오는 보드로 개발을 진행하여, linux kernel에 자신의 코드를 반영하고자 한다면,
  반드시 device tree 기반으로 작업이 이루어져야 함.

장치 트리 기초

- 장치 트리 소스 : arch/$ARCH/boot/dts
- 서드 파티 하드웨어를 이용하는 경우 BSP에 포함되어 제공
- 컴퓨터 시스템의 자원을 나무 같은 계층구조로 결합된 요소의 묶음으로 표현

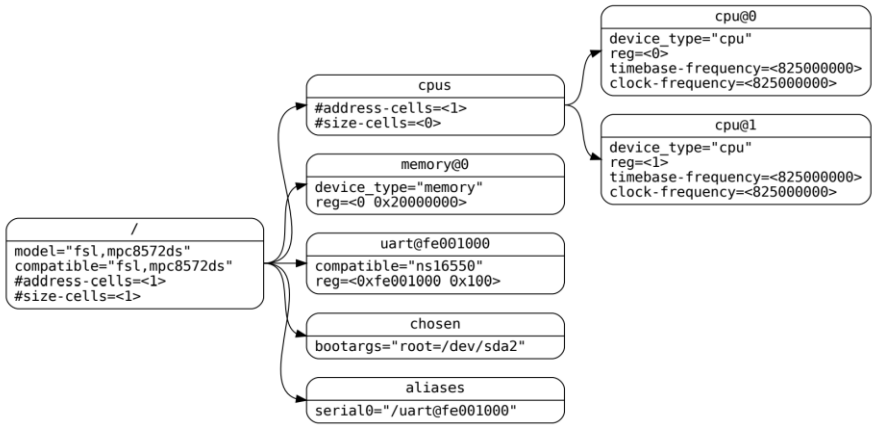
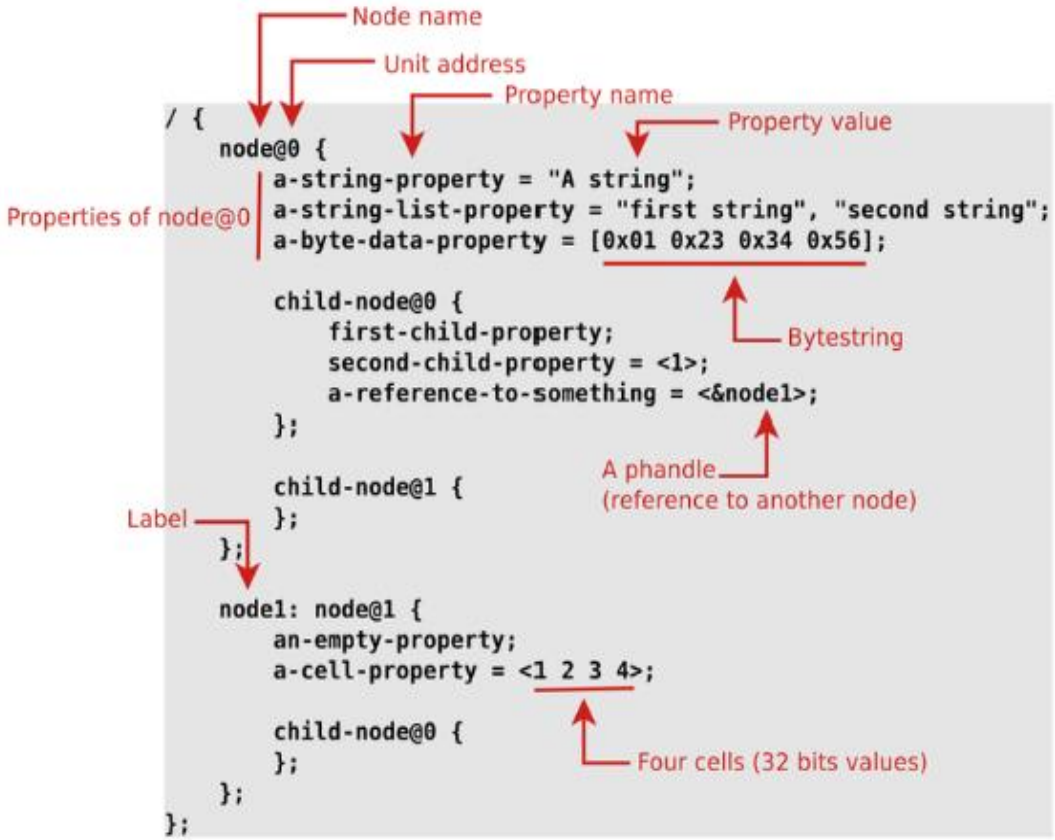


Fig. 2.1: Devicetree Example

devicetree-specification-v0.4-rc1.pdf

장치 트리 기초

- 시스템의 하드웨어 기술(CPU명, 메모리 구성, 장치 명 등)
- 하드웨어 구성을 노드의 계층으로 표현
- 각 노드는 속성과 하위 노드를 포함
- 속성의 이름 : 바이트의 배열(문자열, 숫자, 바이트 등)
- 노드와 속성의 위치를 경로로 표현 (구분자로 ‘/’ 이용)
- 파일명 확장자 : .dts (소스), .dtb(바이너리)



〈참조〉 https://events.static.linuxfound.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf

장치 트리 예제

```
/dts-v1/;
/{
   model = "TI AM335x BeagleBone";
   compatible = "ti,am33xx";
   #address-cells = <1>;
   #size-cells = <1>;
   cpus {
      #address-cells = <1>;
      #size-cells = <0>;
      cpu@0 {
         compatible = "arm,cortex-a8";
         device_type = "cpu";
         reg = <0>;
      };
   };
   memory@0x80000000 {
      device_type = "memory";
      reg = <0x80000000 0x20000000>; /* 512 MB */
   };
};
```

32비트 경우

```
/{
   #address-cells = <2>;
   #size-cells = <2>;
   memory@80000000 {
      device_type = "memory";
      reg = <0x00000000  0x80000000  0  0x80000000>;
   };
};
```

cell의 개수 선언

64비트 경우

장치 트리 – 프로퍼티

- compatible 속성 : 장치 드라이버 (제조사명,장치이름)
- device_type 속성 : 장치의 종류
- reg 속성 : 레지스터 공간에 있는 구성 단위의 범위, 형식 :〈시작주소, 크기, …〉
  노드에 할당된 시작 주소와 길이 의미
- interrupt 속성 :
  - interrupt-controller – 인터럽트 신호를 수신하는 장치로 노드를 선언하는 빈 속성
  - #interrupt-cells – 인터럽트 컨트롤러 노드의 속성. 해당 인터럽트 컨트롤러의 인터럽트 지정자에 몇 개의 셀이 있는지 표시 (#address-cell 및 #size-cell과 유사).
  - interrupt-parent – 연결된 인터럽트 컨트롤러에 대한 팬들을 포함하는 장치 노드의 속성. interrupt-parent 속성이 없는 노드는 부모 노드에서 속성을 상속 할 수도 있음
  - interrupts – 디바이스의 각 인터럽트 출력 신호마다 하나씩 인터럽트 지정자 목록을 포함하는 디바이스 노드의 특성

장치 트리 – 프로퍼티

- **〈참조 사이트〉**
  https://wikidocs.net/3205#part1.2
  https://jung-max.github.io/2019/10/22/Device_Tree_%EB%AC%B8%EB%B2%95/

Device Tree 사용 예

- 라즈베리파이

```
$ fdtdump /boot/firmware/overlays/gpio-led.dtbo

$ vi /boot/firmware/config.txt
    …
   dtoverlay=gpio-led,gpio=18,label=####

$ sudo reboot

$ ls /sys/class/leds/####
$ echo  1 〉/sys/class/leds/####/brightness
$ echo  0 〉/sys/class/leds/####/brightness
```

〈참조〉 https://www.raspberrypi.com/documentation/computers/config_txt.html

u-boot

- 임베디드 분야의 다양한 CPU 아키텍처 및 장치를 지원하는 부트로더
- Denx Software Engineering 에서 유지보수
- 문서 사이트 : https://docs.u-boot.org/en/latest/

- ARM 기반 & 임베디드 Linux 시스템 경우 주로 u-boot를 사용
- u-boot를 기반으로 자신의 Board에 맞게 porting 진행

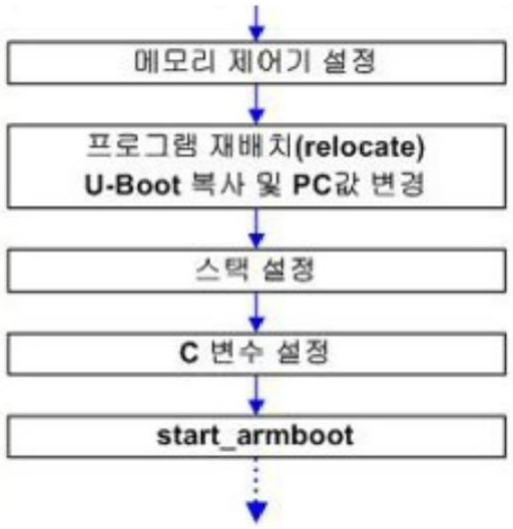
u-boot

- u-boot 소스코드의 doc/README.* 파일 참조

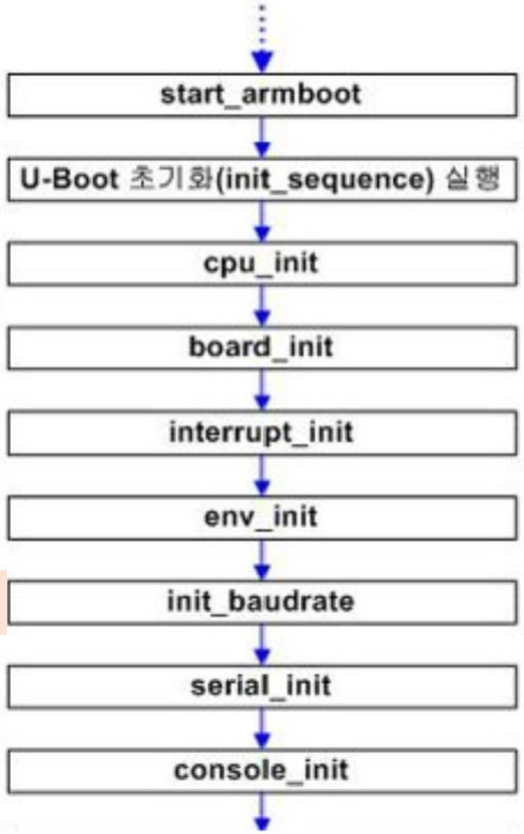
```
$ git clone git://git.denx.de/u-boot.git
$ cd u-boot
$  make  〈board_name〉_defconfig        https://docs.u-boot.org/en/latest/board/index.html
$  make menuconfig
```

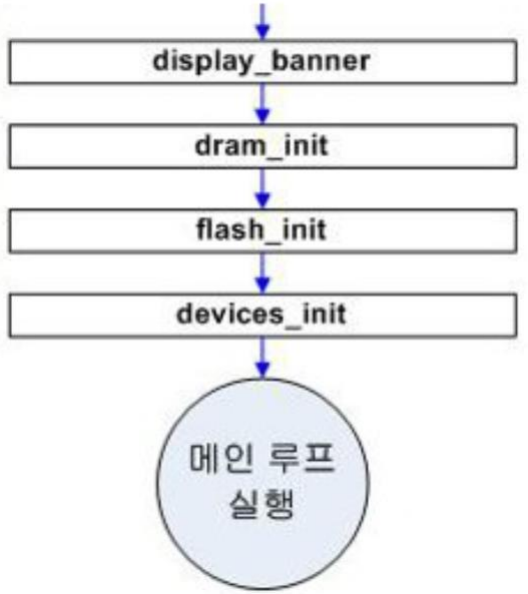
u-boot 초기화

arch/arm/cpu/armv7m/start.S



arch/arm/cpu/armv7m/start.S





u-boot 초기화

- 초기화 코드 순서

u-boot.lds ENTRY(_start)
arch/arm/lib/vectors.S _start
arch/arm/cpu/armv7/start.S reset
arch/arm/lib/crt0.S _main
common/board_r.c board_init_r
common/board_r.c init_sequence_r
common/board_r.c run_main_loop
common/main.c main_loop
common/autoboot.c autoboot_command
common/cli.c run_command_list
common/cli_simple.c cli_simple_run_command_list
common/cli_simple.c cli_simple_run_command
common/command.c cmd_process
common/command.c cmd_call

〈참고〉 https://engineer-rew.tistory.com/entry/U-Boot-Sequence-%EB%B0%8F-%EA%B5%AC%EC%A1%B0-%EB%B6%84%EC%84%9D

u-boot

- 빌드 후 생성되는 파일

  u-boot.map: symbol map
  u-boot: ELF 형식의 U-Boot 이미지 파일
  u-boot.bin: 플래시 메모리에 탑재되는 순수 바이너리 이미지 파일
  u-boot.lds: lds 파일은 링커에게 코드 배치에 대한 방식을 어떻게 할지에 대해 기재한 파일

u-boot 빌드

- 빌드에 필요한 패키지 설치

```
$ sudo apt-get install autoconf build-essential libavahi-client-dev \
    libgnutls28-dev libkrb5-dev libnss-mdns libpam-dev \
    libsystemd-dev libusb-1.0-0-dev zlib1g-dev
```

- u-boot 빌드

```
$ cd u-boot
$ make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu-  rpi_arm64_defconfig
$ make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu-  all
```

〈참조〉 https://dowkim10.tistory.com/3

https://velog.io/@kgh9959/linux-BSP-3

https://pandysong.github.io/blog/post/run_u-boot_in_qemu/

u-boot 설치

- u-boot.bin을 라즈베리파이로 복사
- 라즈베리파이에서

```
$ sudo cp u-boot.bin /boot/firmware/
$ sudo vim /boot/firmware/config.txt
  → 마지막 줄에 추가     kernel=u-boot.bin
$ sudo reboot
```

〈참조〉 임베디드 리눅스에서 U-Boot 부트로더의 이해와 설정 방법
(https://ko.ittrip.xyz/c/embedded-linux-u-boot-setup)

uboot in qemu

- uboot 빌드

```
export CROSS_COMPILE=aarch64-linux-gnu-
make qemu_arm64_defconfig
make
```

- uboot의 bios로 uboot 실행

```
qemu-system-aarch64 -nographic -no-reboot -machine virt -cpu cortex-a57
-bios u-boot.bin
```

```
qemu-system-aarch64 -machine help
```