

# 개발자 패키지로 크로스 컴파일하는 방법

## 내용


- 1 기사 목적
- 2 전제 조건
- 3 Linux 커널 구성 수정
  - 3.1 서문
  - 3.2 간단한 예
- 4 Linux 커널 장치 트리 수정
- 5 내장 Linux 커널 장치 드라이버 수정
- 6 외부 Linux 커널 모듈 수정 / 추가
  - 6.1 외부 인트 리 리눅스 커널 모듈 수정
  - 6.2 외부 트리 외부 Linux 커널 모듈 추가
- 7 U- 부팅 수정
- 8 TF-A 수정
- 9 "hello world"사용자 공간 예제 추가
  - 9.1 소스 코드 파일
  - 9.2 크로스 컴파일
    - 9.2.1 명령 줄
    - 9.2.2 메이크 파일 기반 프로젝트
    - 9.2.3 Autotools 기반 프로젝트
  - 9.3 선내 배포 및 실행
- 10 가지 팁
  - 10.1 마운팅 포인트 생성

## 1 기사 목적

이 기사에서는 **SDK**를 사용한 크로스 컴파일을 설명하는 OpenSTLinux 배포의 개발자 패키지에 대한 간단한 예제를 제공합니다 .

- 소스 코드로 제공되는 소프트웨어 요소 수정 (예 : Linux 커널)
- 소프트웨어 추가 (예 : Linux 커널 모듈 또는 사용자 공간 응용 프로그램)

이 예제는 또한 호스트 컴퓨터에 대한 네트워크 연결을 통해 대상에 크로스 컴파일 결과를 배포하는 방법을 보여줍니다.


 동일한 결과를 얻는 방법에는 여러 가지가 있습니다. 이 기사는 예제 당 하나 이상의 솔루션을 제공하는 것을 목표로합니다. 개발 제약 조건에보다 적합한 다른 방법을 자유롭게 탐색 할 수 있습니다.


## 전제 조건 2 개

OpenSTLinux SDK를 사용한 **크로스 컴파일** 기사 의 전제 조건을 실행해야하며 해당 기사에 설명 된대로 소프트웨어의 크로스 컴파일 및 배치가 알려져 있습니다.

보드와 호스트 시스템은 이더넷 링크를 통해 연결되며 호스트 시스템에서 원격 터미널 프로그램이 시작됩니다 . **터미널을 얻는 방법**을 참조하십시오 .


대상이 시작되고 해당 IP 주소 (<board ip address>)가 알려져 있습니다.

 이 기사의 명령에서 문제점이 발생하면 Linux 커널의 U-Boot 및 TF-A 설치 디렉토리에 있는 README.HOW\_TO.txt 헬퍼 파일 **이** 빌드 참조 임을 기억 하십시오.

 Linux 커널 예제와 관련하여 Linux 커널 은 소스 코드 디렉토리 ( <Linux 커널>)와 다른 전용 빌드 디렉토리 ( 이 페이지 뒷부분의 <Linux 커널 빌드 디렉토리>) 에서 처음으로 설정, 구성 및 빌드 된 것으로 간주됩니다 **소스 디렉토리** ).

## 3 Linux 커널 구성 수정

### 3.1 서문

 **Linux 커널 구성을 수정하기 전에주의해서 읽고 다음 사항에주의하십시오.**

수정하려는 Linux 커널 구성 옵션은 외부 트리 외부 Linux 커널 모듈 (예 : GPU 커널 드라이버)에 의해 사용될 수 있으므로 다시 컴파일해야 합니다. 이러한 모듈은 정의상 커널 트리 구조 외부에 있으며 개발자 패키지 소스 코드로 제공되지 않습니다. 개발자 패키지로 다시 컴파일 할 수 없습니다. 결과적으로 옵션으로 Linux 커널을 재구성하고 다시 컴파일 한 후 보드에 배치하면 외부 트리 외부 Linux 커널 모듈이 더 이상로드되지 않을 수 있습니다.

두 가지 가능한 상황이 있습니다.

- 현재 작업중인 사용 사례에는 문제가되지 않습니다. 이 경우 개발자 패키지를 사용하여 Linux 커널을 수정하고 다시 컴파일 할 수 있습니다.
- 이것은 현재 작업중인 사용 사례의 문제입니다. 이 경우 STM32MP1 배포 패키지를 켜고 Linux 커널 구성을 수정 한 후 이를 사용하여 전체 이미지 (Linux 커널뿐만 아니라 외부 트리 외부 Linux 커널 모듈)를 다시 작성하십시오. ).

예:

- fttrace Linux 커널 기능 을 설치하기 위해 FUNCTION\_TRACER 및 FUNCTION\_GRAPH\_TRACER 옵션이 활성화되었다고 가정 합니다
- 이 기능은 외부 트리 외부 Linux 커널 모듈을 포함하여 전체 커널에 추적 프로그램을 추가하는 데 사용됩니다.

1. 개발자 패키지는 Linux 커널을 재구성 및 재 컴파일하고 보드에 배포하는 데 사용됩니다.

1. 외부 트리 외부 Linux 커널 모듈은 재 컴파일되지 않습니다. GPU 커널 드라이버 의 경우
2. 결과적으로 Linux 커널은 GPU 커널 드라이버 모듈 을로드하지 못합니다 . 그러나 디스플레이가 더 이상 작동하지 않더라도 Linux 커널 부팅에 성공하고 예를 들어 이더넷 또는 USB 연결과 관련된 사용 사례를 디버깅하기에 충분합니다.

2. 배포 패키지는 Linux 커널을 재구성하고 보드에 전체 이미지를 재구성하고 배포하는 데 사용됩니다.

1. GPU 커널 드라이버를 포함하여 외부 트리 외부 Linux 커널 모듈이 재 컴파일됩니다
2. 결과적으로 Linux 커널은 GPU 커널 드라이버 모듈을 성공적으로로드합니다 . 디스플레이가 사용 가능합니다.

### 3.2 간단한 예

이 간단한 예는 연속 메모리 영역 (CMA) 크기에 대해 정의 된 값을 수정합니다.

- 대상 부트 로그의 분석을 통해 CMA 크기 (여기서는 128MB)의 현재 값을 가져옵니다.

```
보드 $> dmesg | grep -i cma
```

STM32MP157C-EV1

```
[0.000000] cma : 0xf0000000에서 128MiB 예약
```

STM32MP157C-DK2

```
[0.000000] cma : 0xd4000000에서 128MiB 예약
```

- <Linux 커널 빌드 디렉토리>로 이동하십시오.

```
PC $> cd <Linux 커널 빌드 디렉토리>
```

- Linux 커널 구성 메뉴를 시작하십시오. Menuconfig 또는 커널 구성 방법을 참조하십시오.
- "장치 드라이버-일반 드라이버 옵션"으로 이동
  - "메가 바이트 크기"를 선택하십시오
  - 값을 256으로 수정
  - 새 구성을 종료하고 저장하십시오.
- 구성 파일 (.config)이 수정 되었는지 확인

```
PC $> grep -i CONFIG_CMA_SIZE_MBYTES .config
CONFIG_CMA_SIZE_MBYTES = 256
```

- Linux 커널 크로스 컴파일 : Menuconfig 또는 커널 구성 방법을 참조하십시오.
- 온보드 Linux 커널 이미지 업데이트 : Menuconfig 또는 커널 구성 방법을 참조하십시오.
- 보드를 재부팅합니다 : Menuconfig 또는 커널 구성 방법을 참조하십시오.
- 대상 부트 로그 분석을 통해 CMA 크기 (256MB)의 새로운 값을 얻습니다.

```
보드 $> dmesg | grep -i cma
```

STM32MP157C-EV1

```
[0.000000] CMA : 소유 256 0xe0000000에서 MiB 크기
```

STM32MP157C-DK2

```
[0.000000] CMA : 소유 256 0xcc000000에서 MiB 크기
```

## 4 Linux 커널 장치 트리 수정

이 간단한 예는 사용자 LED 의 기본 상태를 수정합니다 .

- 보드가 시작된 상태에서; 사용자 녹색 LED (STM32MP157C-EV1의 LD3, STM32MP157C-DK2의 LD5)가 비활성화되어 있는지 확인하십시오.
- <Linux 커널 소스 디렉토리>로 이동하십시오.

```
PC $> cd <Linux 커널 소스 디렉토리>
```

STM32P157C-EV1

- 편집 *아치 / 팔 / 부팅 / DTS / stm32mp157c-ed1.dts*의 장치 트리 소스 파일
- 아래에 강조 표시된 줄을 추가하십시오

```
주도 {
    호환 가능 = "gpio-leds";
    푸른 {
        레이블 = "하트 비트";
        gpios = <& gpiod 9 GPIO _ACTIVE_HIGH>;
        linux, default-trigger = "하트 비트";
        기본 상태 = "off";
    };
    녹색 {
        레이블 = "stm32mp : green : user";
        gpios = <& gpioa 14 GPIO _ACTIVE_LOW>;
        기본 상태 = "on";
    };
};
```

STM32MP157C-DK2

- 편집 *아치 / 팔 / 부팅 / DTS / stm32mp157a-dk1.dts*의 장치 트리 소스 파일
- 아래에 강조 표시된 줄을 추가하십시오

```
주도 {
    호환 가능 = "gpio-leds";
    푸른 {
        레이블 = "하트 비트";
        gpios = <& gpiod 11 GPIO _ACTIVE_HIGH>;
        linux, default-trigger = "하트 비트";
        기본 상태 = "off";
    };
    녹색 {
        레이블 = "stm32mp : green : user";
        gpios = <& gpioa 14 GPIO _ACTIVE_LOW>;
        기본 상태 = "on";
    };
};
```

- <Linux 커널 빌드 디렉토리>로 이동하십시오.

```
PC $> cd <Linux 커널 빌드 디렉토리>
```

- 장치 트리 Blob 생성 (\*.dtb)

```
PC $> make dtbs
PC $> cp arch / arm / boot / dts / stm32mp157 * .dtb install_artifact / boot /
```

- 보드의 장치 트리 모양 업데이트

```
PC $> scp install_artifact / boot / stm32mp157 * .dtb root @ <보드 IP 주소> : / boot /
```



는 IF / 부팅 설치 지점이 아직 존재하지 않는, 참조하시기 바랍니다 장치 지점을 만드는 방법

- 보드를 재부팅

```
보드 $> 재부팅
```

- 사용자 녹색 LED (STM32MP157C-EV1의 LD3, STM32MP157C-DK2의 LD5)가 활성화되어 있는지 확인하십시오 (녹색)

## 5 내장 Linux 커널 장치 드라이버 수정

이 간단한 예는 디스플레이 드라이버가 프로브 될 때 무조건 로그 정보를 추가합니다.

- 디스플레이 드라이버가 검사 될 때 로그 정보가 없는지 확인

```
보드 $> dmesg | grep -i stm_drm_platform_probe
보드 $>
```

- <Linux 커널 소스 디렉토리>로 이동하십시오.

```
PC $> cd <Linux 커널 소스 디렉토리>
```

- ./drivers/gpu/drm/stm/drv.c 소스 파일을 편집 하십시오.
- stm\_drm\_platform\_probe 함수 에 로그 정보 추가

```
정적 int stm_drm_platform_probe (struct platform_device * pdev)
{
    struct device * dev = &pdev-> dev;
    struct drm_device * ddev;
    ret;
    [...]

    DRM _INFO ( "간단한 예-% s \ n", __func__);

    리턴 0;
    [...]
}
```

- <Linux 커널 빌드 디렉토리>로 이동하십시오.

```
PC $> cd <Linux 커널 빌드 디렉토리>
```

- Linux 커널 크로스 컴파일 ( README.HOW\_TO.txt 도우미 파일 에서로드 주소 확인 )

```
PC $> make uImage LOADADDR = 0xC2000040
PC $> cp arch / arm / boot / uImage install_artifact / boot /
```

- 온보드 Linux 커널 이미지 업데이트

```
PC $> scp install_artifact / boot / u 이미지 root @ <보드 IP 주소> : / boot /
```



는 IF / 부팅 설치 지점이 아직 존재하지 않는, 참조하시기 바랍니다 **장치 지점을 만드는 방법**

- 보드를 재부팅

```
보드 $> 재부팅
```

- 디스플레이 드라이버가 프로브 될 때 로그 정보가 있는지 확인하십시오.

```
보드 $> dmesg | grep -i stm_drm_platform_probe
[2.995125] [drm] 간단한 예 -stm_drm_platform_probe
```

## 6 외부 Linux 커널 모듈 수정 / 추가

Linux 커널의 대부분의 장치 드라이버 (모듈)는 커널 자체 (내장 / 내부 모듈) 또는 / 아래의 루트 파일 시스템에 배치해야하는로드 가능한 커널 모듈 (LKM / 외부 모듈)로 컴파일 할 수 있습니다. lib / modules 디렉토리 외부 모듈은 트리 내부 (커널 트리 구조) 또는 트리 외부 (커널 트리 구조 외부) 일 수 있습니다.

### 6.1 외부 인트 리 리눅스 커널 모듈 수정

이 간단한 예는 가상 비디오 테스트 드라이버 (vivid) 커널 모듈이 프로브되거나 제거 될 때 무조건 로그 정보를 추가합니다.

- <Linux 커널 소스 디렉토리>로 이동하십시오.

```
PC $> cd <Linux 커널 소스 디렉토리>
```

- ./drivers/media/platform/vivid/vivid-core.c 소스 파일을 편집 하십시오.
- vivid\_probe 및 vivid\_remove 함수 에 로그 정보 추가

```
static int vivid_probe (struct platform_device * pdev)
{
    const struct font_desc * font = find_font ( "VGA8x16");
    int ret = 0, i;
    [...]

    / * n_devs는 할당 된 실제 장치 수를 반영합니다 * /
    n_devs = i;

    pr_info ( "간단한 예-% s \ n", __func__ ) ;

    리트 윗을 반환;
}
```

```
static int vivid_remove (struct platform_device * pdev)
{
    struct vivid_dev * dev;
    부호없는 int i, j;
    [...]

    pr_info ( "간단한 예-% s \ n", __func__ ) ;

    리턴 0;
}
```

- <Linux 커널 빌드 디렉토리>로 이동하십시오.

```
PC $> cd <Linux 커널 빌드 디렉토리>
```

- 리눅스 커널 모듈 크로스 컴파일

```
PC $> make 모듈
PC $> make INSTALL_MOD_PATH = "./ install_artifact"modules_install
```

- 보드의 생생한 커널 모듈 업데이트 (커널 버전 <커널 버전>을 확인하십시오)

```
PC $> scp install_artifact / lib / modules / <커널 버전> /kernel/drivers/media/platform/vivid/vivid.ko root @ <보드 IP 주소> :
```

또는

```
PC $> scp -r install_artifact / lib / modules / * root @ <보드 IP 주소> : / lib / modules /
```

- 로드 가능한 커널 모듈에 대한 종속성 설명을 업데이트하고 디스크의 데이터를 메모리와 동기화

```
보드 $> / sbin / depmod -a
보드 $> 동기화
```

- 생생한 커널 모듈을 Linux 커널에 삽입

```
보드 $> 모드 프로브 생생한
[...]
```

[3412.784638] 간단한 예 -vivid\_probe

- Linux 커널에서 생생한 커널 모듈 제거

```
보드 $> rmmod 생생한
[...]
```

[3423.708517] 간단한 예 -vivid\_remove

## 6.2 외부 트리 외부 Linux 커널 모듈 추가

이 간단한 예제는 "Hello World"외부 트리 외부 Linux 커널 모듈을 Linux 커널에 추가합니다.

- 전제 조건 : Linux 소스 코드가 설치되어 있고 Linux 커널이 크로스 컴파일되었습니다.
- 모든 소스 코드를 포함하는 작업 디렉토리 (즉, Linux 커널, U-Boot 및 TF-A 소스 코드 디렉토리를 포함하는 디렉토리)로 이동하십시오.

```
PC $> cd <tag> / sources / arm- <distro> -linux-gnueabi
```

- Linux 커널 소스 코드와 구성 파일 (.config) 이 모두 포함 된 Linux 커널 빌드 디렉토리의 경로를 KERNEL\_SRC\_PATH로 내보내십시오.

```
PC $> 내보내기 KERNEL_SRC_PATH = $ PWD / <Linux 커널 빌드 디렉토리> /
```

예:

```
PC $> 내보내기 KERNEL_SRC_PATH = $ PWD / linux-stm32mp-4.19-r0 / build
```

- 이 커널 모듈 예제의 디렉토리를 만듭니다

```
PC $> mkdir kernel_module_example
PC $> cd 커널 _module_example
```

- 이 커널 모듈 예제의 소스 코드 파일을 작성하십시오. *kernel\_module\_example.c*

```
// SPDX 식별자 : GPL-2.0
/ *
 * 저작권 (C) STMicroelectronics SA 2018
 *
 * 저자 : Jean-Christophe Trotin <jean-christophe.trotin@st.com>
 *
 * /

모든 커널 모듈에 대해 #include <linux / module.h> / * * /
KERN_INFO의 경우 #include <linux / kernel.h> / * * /
__init 및 __exit 매크로의 경우 #include <linux / init.h> / * * /

static int __init kernel_module_example_init (void)
{
    printk (KERN_INFO "커널 모듈 예 : Hello World from STMicroelectronics \ n");
    리턴 0;
}

정적 무효 __exit kernel_module_example_exit (void)
```

- 이 커널 모듈 예제에 대한 makefile을 작성하십시오. *Makefile*



makefile의 모든 들여 쓰기는 표입니다

```
# 외부의 외부 트리 외부 리눅스 커널 모듈 예제를위한 Makefile

# 빌드 할 오브젝트 파일
obj-m := kernel_module_example.o

# 리눅스 커널 소스 코드를 포함하는 디렉토리 경로
# 및 구성 파일 (.config)
KERNEL_DIR? = $ (KERNEL_SRC_PATH)

# 생성 된 객체를 포함하는 디렉토리 경로
DESTDIR? = $ (KERNEL_DIR) / install_artifact

# 컴파일 할 소스 파일이 들어있는 디렉토리 경로
PWD := $ (셸 암호)

기본:
    $ (MAKE) -C $ (KERNEL_DIR) M = $ (PWD) 모듈

설치:
```

- 커널 모듈 예제 크로스 컴파일

```
PC $> 깨끗한
PC 만들기 $> PC 만들기
$> 설치하기
```

- <Linux 커널 빌드 디렉토리>로 이동하십시오.

```
PC $> cd <Linux 커널 빌드 디렉토리>
```

- 생성 된 커널 모듈 예제는 *install\_artifact / lib / modules / <kernel version> / extra/kernel\_module\_example.ko*에 있습니다.
- 이 커널 모듈 예제를 보드에 푸시하십시오 (커널 버전 <커널 버전>을 확인하십시오)

```
PC $> ssh root @ <보드 IP 주소> mkdir -p / lib / modules / <커널 버전> / 추가
PC $> scp install_artifact / lib / modules / <커널 버전> /extra/kernel_module_example.ko root @ <보드 IP address> : / lib / m
```

또는

```
PC $> scp -r install_artifact / lib / modules / * root @ <보드 IP 주소> : / lib / modules /
```

- 로드 가능한 커널 모듈에 대한 종속성 설명을 업데이트하고 디스크의 데이터를 메모리와 동기화

```
보드 $> / sbin / depmod -a
보드 $> 동기화
```

- 커널 모듈 예제를 Linux 커널에 삽입

```
보드 $> modprobe kernel_module_example
[18167.821725] 커널 모듈 예 : STMicroelectronics의 hello world
```

- Linux 커널에서 커널 모듈 예제를 제거하십시오.

```
보드 $> rmmod kernel_module_example
[18180.086722] 커널 모듈 예 : STMicroelectronics의 작별
```

## 7 U-Boot 수정

이 간단한 예는 U-Boot 가 시작될 때 무조건 로그 정보를 추가 합니다. 의 범위 내에서 신뢰할 수있는 부팅 체인 , U-부팅은 두 번째 단계의 부트 로더 (로 사용되니다 SSBL ).

- 상기 모양이 U-부트 로그 정보를 때 보드 재부팅

```
보드 $> 재부팅
```

### STM32MP157C-EV1

```
[...]
U- 부트 2018.11-stm32mp-r2

CPU : STM32MP157CAA 개정 B
모델 : STMicroelectronics STM32MP157C 평가판 평가판 평가판
보드 : 트러스트 모드의 stm32mp1 (st, stm32mp157c-ev1)
[...]
```

### STM32MP157C-DK2

```
[...]
U- 부트 2018.11-stm32mp-r2

CPU : STM32MP157CAC 개정 B
모델 : STMicroelectronics STM32MP157C-DK2 디스커버리 보드
보드 : 트러스트 모드의 stm32mp1 (st, stm32mp157c-dk2)
[...]
```

- < U-Boot 소스 디렉토리>로 이동하십시오

```
PC $> cd < U-Boot 소스 디렉토리>
```



예 :  
**PC \$>** cd u-boot-stm32mp-2018.11-r0 / u-boot-2018.11

- ./board/st/stm32mp1/stm32mp1.c 소스 파일을 편집 하십시오.
- *바독판* 기능 에 로그 정보 추가

```
int checkboard (void)
{
    문자 * 모드;

    [...]
    쉼 ( "\ n");
    printf ( " U-Boot 간단한 예 \ n");
    [...]
}
```

```
리턴 0;
}
```

- 다음 명령으로 지원되는 구성 목록을 가져옵니다.

**PC \$>** make -f \$ PWD ../ Makefile.sdk 도움말

- U- 부트 크로스 컴파일 : STM32MP157C-EV1 및 STM32MP157C-DK2 용 신뢰할 수있는 부팅

**PC \$>** make -f \$ PWD ../ Makefile.sdk all UB00T\_CONFIGS = stm32mp15\_trusted\_defconfig, trusted, u-boot.stm32

- 컴파일 결과가 저장된 디렉토리로 이동


**PC \$>** cd 빌드-신뢰 /

- 보드를 재부팅하고 U-boot 셸에서 멈추려면 아무 키나 누르십시오.

**보드 \$>** 재부팅  
 [...]
 자동 부팅을 중지하려면 아무 키나 누르십시오 : 0  
 STM32MP>

- USB OTG 포트 를 통해 호스트 시스템과 보드 사이에 USB 케이블을 연결하십시오
- 에서 U-부트 셸, USB 대용량 저장 함수를 호출

STM32MP> 음 0 mmc 0

 U-Boot UMS 기능 사용에 대한 자세한 내용은 U-Boot 에서 USB 대용량 저장소를 사용하는 방법을 참조하십시오.

- 호스트 시스템에서 2 차 스테이지 부트 로더 ( *ssbl* ) 와 연관된 파티션을 확인하십시오 . *sd3* here

```
PC $> ls -l / dev / disk / by-partlabel /
총 0
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 08:57 bootfs-> ../../sdb4
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 08:57 fsbl1-> ../../sdb1
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 08:57 fsbl2-> ../../sdb2
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 08:57 rootfs-> ../../sdb6
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 08:57 ssbl-> ../../sdb3
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 08:57 userfs-> ../../sdb7
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 08:57 vendorfs-> ../../sdb5
```

- U-Boot 바이너리를 전용 파티션에 복사

## STM32MP157C-EV1

```
PC $> dd if = u-boot-stm32mp157c-ev1-trusted.stm32 of = / dev / sdb3 bs = 1M conv = fdatasync
```

## STM32MP157C-DK2

```
PC $> dd if = u-boot-stm32mp157c-dk2-trusted.stm32 of = / dev / sdb3 bs = 1M conv = fdatasync
```

- U-Boot 셀 재설정

STM32MP&gt; 리셋

- 보드가 재부팅 될 때 새로운 U-Boot 로그 정보를 확인하십시오

## STM32MP157C-EV1

```
[...]
U- 부트 2018.11-stm32mp-r2

CPU : STM32MP157CAA 개정 B
모델 : STMicroelectronics STM32MP157C 평가판 평가판 평가판
보드 : 트러스트 모드의 stm32mp1 (st, stm32mp157c-ev1)
U-Boot 간단한 예
[...]
```

## STM32MP157C-DK2

```
[...]
U- 부트 2018.11-stm32mp-r2

CPU : STM32MP157CAC 개정 B
모델 : STMicroelectronics STM32MP157C-DK2 디스커버리 보드
보드 : 트러스트 모드의 stm32mp1 (st, stm32mp157c-dk2)
U-Boot 간단한 예
[...]
```

## 8 TF-A 수정

이 간단한 예는 TF-A 가 시작될 때 무조건 로그 정보를 추가 합니다. 의 범위 내에서 신뢰할 수 있는 부트 체인 , TF-A는 첫 번째 단계의 부트 로더 (로 사용 FSBL ).

- 보드가 재부팅 될 때 TF-A 로그 정보를 살펴보세요

```
보드 $> 재부팅
[...]
정보 : MPU (MPSYSRST)에 의해 생성 된 시스템 재설정
정보 : SDRAM 사용
[...]
```

- < TF-A 소스 디렉토리>로 이동하십시오

```
PC $> cd < TF-A 소스 디렉토리>
```

```
예 :
PC $> cd tf-a-stm32mp-2.0-r0 / arm-trusted-firmware-2.0
```

- ./plat/st/stm32mp1/bl2\_plat\_setup.c 소스 파일을 편집 하십시오.

- `print_reset_reason` 함수에 로그 정보 추가

```
static void print_reset_reason (void)
{
    [...]
    INFO ( "리셋 이유 (0x % x) : \n", rstsr);

    INFO ( " TF- 간단한 예제 \n");
    [...]
}
```

- 다음 명령으로 지원되는 구성 목록을 가져옵니다.

```
PC $> make -f $ PWD ../ Makefile.sdk 도움말
```

- TF-A 크로스 컴파일 : STM32MP157C-EV1 및 STM32MP157C-DK2 용 신뢰할 수 있는 부팅

```
PC $> make -f $ PWD ../ Makefile.sdk 모두 TF_A_CONFIG = trusted
```

- 컴파일 결과가 저장된 디렉토리로 이동

```
PC $> cd ../build/trusted
```

- 보드를 재부팅하고 U-boot 셸에서 멈추려면 아무 키나 누르십시오.

```
보드 $> 재부팅
[...]
자동 부팅을 중지하려면 아무 키나 누르십시오 : 0
STM32MP>
```

- USB OTG 포트 를 통해 호스트 시스템과 보드 사이에 USB 케이블을 연결하십시오
- 에서 U-부트 셸, USB 대용량 저장 함수를 호출

```
STM32MP> 음 0 mmc 0
```



U-Boot UMS 기능 사용에 대한 자세한 내용은 U-Boot [에서 USB 대용량 저장소를 사용하는 방법을 참조하십시오.](#)

- 호스트 시스템에서 첫 번째 단계 부트 로더와 연관된 파티션 ( 백업으로 `fsbl1` 및 `fsbl2` ) 과 관련된 파티션을 확인하십시오 . `sdb1` 및 `sdb2` (백업으로) 여기

```
PC $> ls -l / dev / disk / by-partlabel /
총 0
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 10:01 bootfs-> ../../sdb4
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 10:01 fsbl1-> ../../sdb1
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 10:01 fsbl2-> ../../sdb2
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 10:01 rootfs-> ../../sdb6
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 10:01 ssbl-> ../../sdb3
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 10:01 userfs-> ../../sdb7
lrwxrwxrwx 1 루트 루트 10 2 월 8 일 10:01 vendorfs-> ../../sdb5
```

- TF-A 바이너리를 전용 파티션에 복사하십시오 . 새 TF-A 바이너리 를 테스트하려면 백업 FSBL ( `fsbl2` )에 이전 TF-A 바이너리 를 유지하는 것이 유용 할 수 있습니다

STM32MP157C-EV1

```
PC $> dd if = tf-a-stm32mp157c-ev1-trusted.stm32 of = / dev / sdb1 bs = 1M conv = fdatasync
```



권한이 거부 된 경우 / dev / sdb1 에 대한 추가 권한을 설정할 수 있습니다 . `sudo chmod 777 / dev / sdb1`

STM32MP157C-DK2

```
PC $> dd if = tf-a-stm32mp157c-dk2-trusted.stm32 of = / dev / sdb1 bs = 1M conv = fdatasync
```

- U-Boot 셸 재설정

에서 U-부트 셸에서 Ctrl + C 전에 손을 다시 얻을 수 있습니다.

STM32MP> 리셋

- 보드가 재부팅 될 때 새로운 TF-A 로그 정보를 살펴보세요

```
[...]
정보 : MPU (MPSYSRST)에 의해 생성 된 시스템 재설정
정보 :      TF-A 간단한 예
정보 : SDMMC 사용
[...]
```

## 9 "hello world"사용자 공간 예제 추가

OpenSTLinux SDK 덕분에 OpenEmbedded 빌드 시스템 외부에서 프로젝트를 쉽게 개발할 수 있습니다. 이 장에서는 간단한 "hello world"예제를 컴파일하고 실행하는 방법을 보여줍니다.

### 9.1 소스 코드 파일

- 모든 소스 코드가 포함 된 작업 디렉토리 (예 : Linux 커널, U-Boot 및 TF-A 소스 코드 디렉토리가 포함 된 디렉토리)로 이동하십시오.

```
PC $> cd <tag> / sources / arm- <distro> -linux-gnueabi
```

- 이 사용자 공간 예제에 대한 디렉토리를 작성하십시오.

```
PC $> mkdir hello_world_example
PC $> cd hello_world_example
```

- 이 사용자 공간 예에 대한 소스 코드 파일을 작성하십시오. `hello_world_example.c`

```
// SPDX 식별자 : GPL-2.0
/ *
 * 저작권 (C) STMicroelectronics SA 2018
 *
 * 저자 : Jean-Christophe Trotin <jean-christophe.trotin@st.com>
 *
 * /

#include <stdio.h>
#include <unistd.h>

int main (int argc, char ** argv)
{
    int i = 11;

    printf ( "\ n 사용자 공간 예 : STMicroelectronics의 hello world \ n");
    setbuf (stdout, NULL);
    동안 (i--) {
        printf ( "% i", i);
```

## 9.2 크로스 컴파일

OpenSTLinux SDK 를 사용하여 사용자 공간 예제를 크로스 컴파일 하는 세 가지 방법 이 아래에 제안되어 있습니다. (1) 명령 줄 (2) makefile 기반 프로젝트 (3) autotools 기반 프로젝트.

### 9.2.1 명령 줄

이 방법을 사용하면 단일 소스 코드 파일을 신속하게 크로스 컴파일 할 수 있습니다. 프로젝트에 파일이 하나만있는 경우 적용됩니다.

교차 개발 툴체인은 대상 아키텍처에서 실행되는 바이너리를 생성하는 데 필요한 헤더 파일 및 라이브러리가 포함 된 sysroot와 연관되어 있습니다 ( [OpenSTLinux distribution # Native 및 대상 sysroots 용 SDK 참조](#) ).

sysroot 위치는 --sysroot 옵션으로 지정됩니다 .

sysroot 위치는 --sysroot 옵션을 사용하여 지정해야 합니다. CC에 의해 생성 된 환경 변수 SDK는 이미 포함 --sysroot 옵션을 가리 SDK의 SYSROOT 위치.

```
PC $> echo $ CC
arm-openstlinux_weston-linux-gnueabi-gcc -march = armv7ve -mthumb -mcpu = neon-vfpv4 -mfloat-abi = hard -mcpu = cortex-a7 --sysroot = < SDK 설치 디렉토리 > / SDK / sysroots / cortex-a7t2hf-neon-vfpv4-openstlinux-w
```

- 생성 된 바이너리를 저장할 디렉토리를 만듭니다.

```
PC $> mkdir -p install_artifact install_artifact / usr install_artifact / usr / local install_artifact / usr / local / bin
```

- 사용자 공간 예제를위한 단일 소스 코드 파일을 크로스 컴파일

```
PC $> $ CC hello_world_example.c -o ./install_artifact/usr/local/bin/hello_world_example
```

### 9.2.2 메이크 파일 기반 프로젝트

이 방법의 경우, 크로스 툴체인 환경 설정 스크립트를 실행하여 셸립 크로스 툴체인 환경 변수는 일반적인 될 수 있습니다 *메이크업*의 규칙.

예를 들어 다음 환경 변수를 참조하십시오.

```
PC $> echo $ CC
arm-openstlinux_weston-linux-gnueabi-gcc -march = armv7ve -mthumb -mcpu = neon-vfpv4 -mfloat-abi = hard -mcpu = cortex-a7 --sysroot = < SDK 설치 디렉토리 > / SDK / sysroots / cortex-a7t2hf-neon-vfpv4-openstlinux-w
PC $> echo $ CFLAGS
-O2 -pipe -g -feliminate-unused-debug-types
PC $> echo $ LDFLAGS
필요에 따라 -Wl, -O1 -Wl, -해시 스타일 = gnu -Wl, -
PC $> echo $ LD
arm-openstlinux_weston-linux-gnueabi-ld --sysroot = < SDK 설치 디렉토리 > / SDK / sysroots / cortex-a7t2hf-neon-vfpv4-openstlinux-w
```

- 이 사용자 공간 예에 대한 makefile을 작성하십시오. *Makefile*



makefile의 모든 들여 쓰기는 표입니다

```

PROG = hello_world_example
SRCS = hello_world_example.c
OBS = $ (SRCS : .c = .o)

CLEANFILES = $ (PROG)
INSTALL_DIR = ./install_artifact/usr/local/bin

# 필요한 경우 CFLAGS의 추가 / 변경 옵션
# CFLAGS += <새 옵션>

$ (PROG) : $ (OBS)
    $ (CC) $ (CFLAGS) -o $ (PROG) $ (OBS)

.co :
    $ (CC) $ (CFLAGS) -c $ <-o $ @

모두 : $ (PROG)
```

- 프로젝트 크로스 컴파일

```
PC $> make
PC $> make 설치
```

### 9.2.3 Autotools 기반 프로젝트

이 방법은 GNU autotools를 기반으로 프로젝트를 만듭니다.

- 이 사용자 공간 예에 대한 makefile을 작성하십시오. *Makefile.am*

```
bin_PROGRAMS = hello_world_example
hello_world_example_SOURCES = hello_world_example.c
```

- 이 사용자 공간 예에 대한 구성 파일을 작성하십시오. *configure.ac*

```
AC_INIT (hello_world_example, 0.1)
AM_INIT_AUTOMAKE ([외국])
AC_PROG_CC
AC_PROG_INSTALL
AC_OUTPUT (메이크 파일)
```

- 로컬 aclocal.m4 파일을 생성하고 구성 스크립트를 작성하십시오

```
PC $> aclocal
PC $> autoconf
```

- GNU 코딩 표준에 필요한 파일 생성 (규정 준수)

```
PC $> touch 뉴스 읽어보기 저자 변경 내역
```

- SDK 스크립트에 대한 링크 생성

```
PC $> automake -a
```

- 프로젝트 크로스 컴파일

```
PC $> ./configure $ {CONFIGURE_FLAGS}
PC $> make
PC $> make install DESTDIR =. / install_artifact
```

### 9.3 선상에서 배포 및 실행

- 이 사용자 공간 예제에 대해 생성 된 바이너리가 *./install\_artifact/usr/local/bin/hello\_world\_example*에 있는지 확인하십시오 .
- 이 바이너리를 보드에 밀어 넣습니다.

```
PC $> scp -r install_artifact / * root @ <보드 IP 주소> : /
```

- 이 사용자 공간 예제 실행

```
보드 $> cd / usr / local / bin
보드 $> ./hello_world_example
```

```
사용자 공간 예 : STMicroelectronics의 Hello World
10 9 8 6 6 4 3 2 1 0
사용자 공간 예 : STMicroelectronics의 작별
```

## 팁 10 개

### 10.1 마운팅 포인트 생성

부트 파일 시스템 (bootfs 파티션)의 마운트 지점을 만드는 것이 목표입니다.

- 부팅 파일 시스템과 관련된 파티션 레이블을 찾습니다

```
보드 $> ls -l / dev / disk / by-partlabel /
총 0
lrwxrwxrwx 1 루트 루트 15 1 월 23 일 17:00 bootfs-> ../../mmcblk0p4
lrwxrwxrwx 1 루트 루트 15 1 월 23 일 17:00 fsbl1-> ../../mmcblk0p1
lrwxrwxrwx 1 루트 루트 15 1 월 23 일 17:00 fsbl2-> ../../mmcblk0p2
lrwxrwxrwx 1 루트 루트 15 1 월 23 일 17:00 rootfs-> ../../mmcblk0p6
lrwxrwxrwx 1 루트 루트 15 1 월 23 일 17:00 ssbl-> ../../mmcblk0p3
lrwxrwxrwx 1 루트 루트 15 1 월 23 일 17:00 userfs-> ../../mmcblk0p7
lrwxrwxrwx 1 루트 루트 15 1 월 23 일 17:00 vendorfs-> ../../mmcblk0p5
```

- / boot 디렉토리의 / dev / mmcblk0p4 에 있는 부트 파일 시스템을 연결하십시오.

```
보드 $> 마운트 / dev / mmcblk0p4 / boot
```