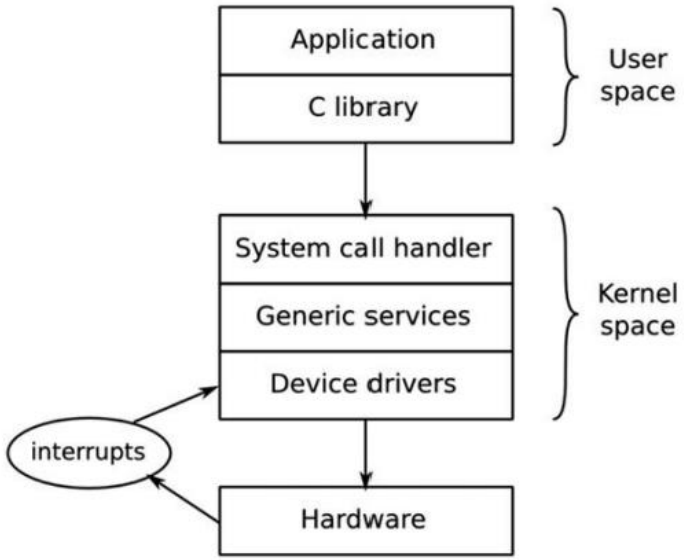
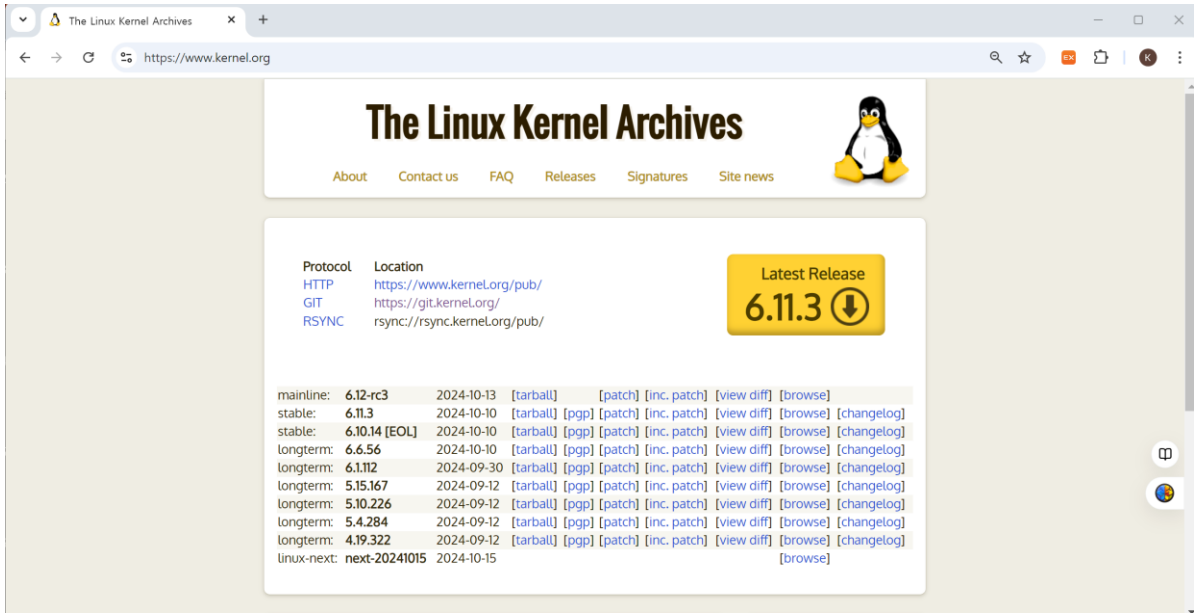
04. 커널 구성과 빌드

- 사용자 공간, 커널 공간

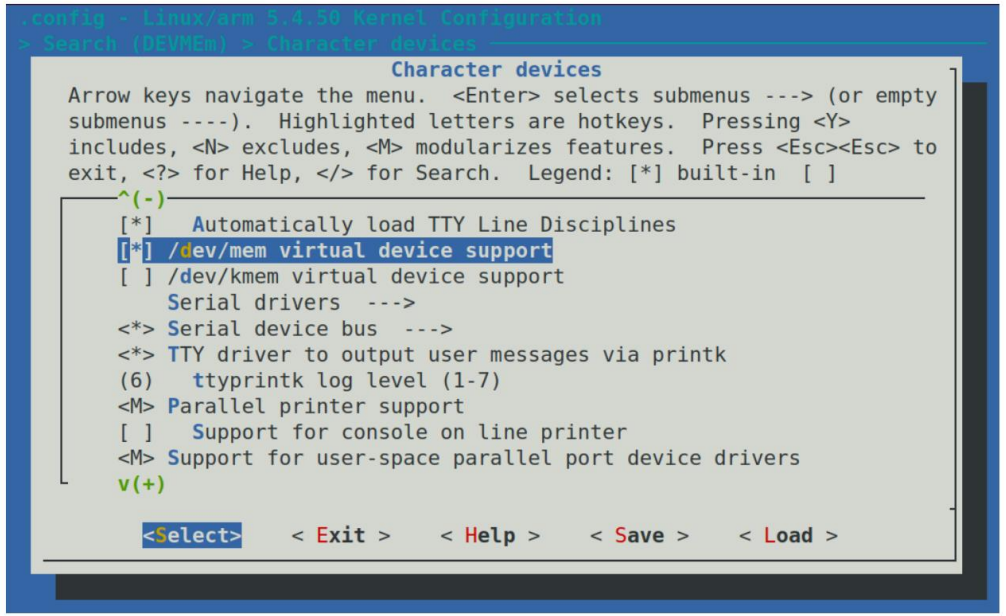


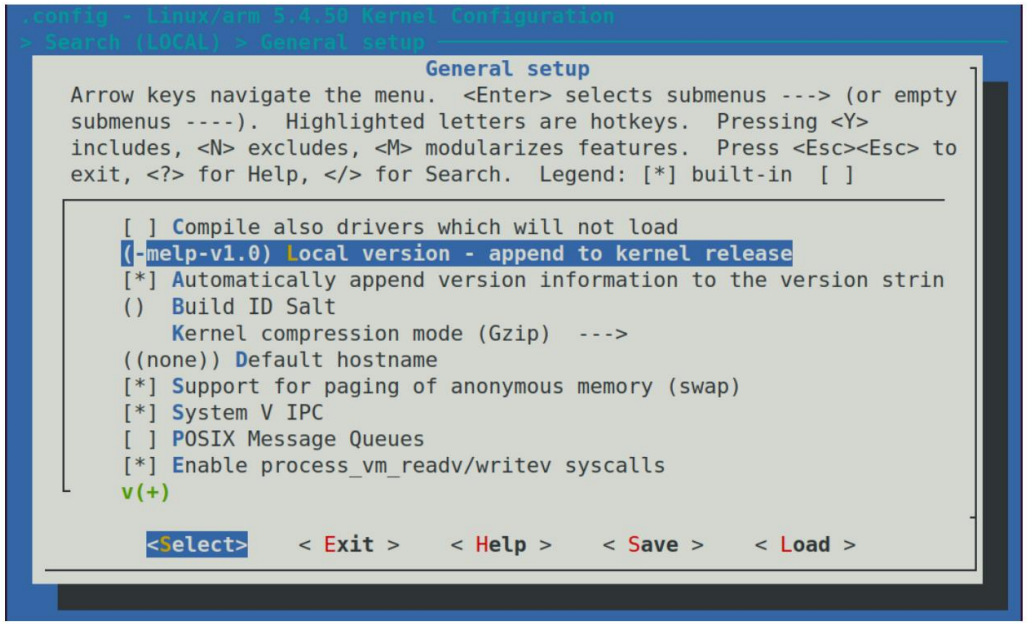
https://www.kernel.org/



커널 구성과 빌드

- 커널 구성 (Kconfig)





$ make ARCH=*** ####_defconfig

커널 구성과 빌드

- 커널 모듈

  : 커널의 기능이나 장치 드라이버 등 다양하게 활용

  커널 버전과 밀접하게 연관

- 임베디드 분야의 장치 드라이버의 경우 일반적으로 모듈보다는 커널에 직접 내재 시키는 경우가 많음.
- 임베디드 시스템에서 모듈 사용이 좋은 경우

  라이선스 등의 이유로 비공개 모듈이 있는 경우
  필수적인 장치 드라이버가 아닌 경우 로딩 시간을 연기함으로써 부팅 시간을 줄 일수 있음
  로드해야하는 드라이버가 여러 개라 정적으로 링크 시 메모리르 많이 차지하게 되는 경우 (예 USB 장치)

커널 구성과 빌드

- 커널 빌드(Kbuild)

  .config 파일로부터 구성정보를 취합하여 의존 관계를 파악하고 커널 이미지를 빌드하는 Makefile 수행

```
obj-y += mem.o random.o                    # 무조건 컴파일하여 커널 이미지에 포함시킴
pbj-$(CONFIG_TTY_PRINTK) += ttyprintk.o    # CONFIG_TTY_PRINTK가 y 면 커널 내장, m 이면
                                           # 모듈로, 정의 되어있지 않으면 컴파일되지 않음
```

- 커널 이미지 빌드 빌드 실패시 V=1 추가하여 다시 실행

```
$ make –j# ARCH=arm CROSS_COMPILE=aarch64-linux-gnu-  Image | zImage | uImage
```

- vmlinux 파일 : ELF 바이너리 (디버깅 정보 포함)

Image : vmlinux + ELF 처리가 가능한 코드 결합
zimage : 압축된 Image 파일 + 압축해제 및 재배치 코드와 결함
uImage : zImage + U-Boot 헤더(64바이트)

커널 구성과 빌드

- 커널 소스 클리닝

clean : .o 파일을 포함 컴파알 도중에 생성된 파일 삭제

mrproper : .config를 비롯한 중간에 생성된 모든 파일 삭제, 소스코드 설치 시 상태로 되돌리기

distclean : mrproper와 유사, 편집기 백업파일, 패치 파일 등 개발 시 임시로 만든 파일들도 모두 삭제

커널 구성과 빌드

- 장치 트리 컴파일(dts)

```
$ make ARCH=arm64 dts
  . . .
  DTC     arch/arm64/boot/dts/alphine-db.dtb
  DTC     arch/arm64/boot/dts/artpec6-devboard.dtb.dtb
  . . .
```

- 모듈 컴파일

```
$ make j=# ARCH=arm64 dts CROSS_COMPILE=aarch64-linux-gnu- modules
  . . .
 $ make j=# ARCH=arm64 dts CROSS_COMPILE=aarch64-linux-gnu-  modules_install
```

커널 구성과 빌드

- 커널 부팅 과정 중 초가화 작업이 이루어진 후 사용자 공간을 제공하기 위해 루트 파일시스템 마운트 (램 디스크 또는 실제 파일시스템)

  커널 소스 코드 중 init/main.c 의 rest_init() 함수 실행
  PID 1의 쓰레드 생성 kernel_init() 함수 실행
  램디스크가 있는 경우 프로그램 /init 실행, 없는 경우 명령 줄의 root= 인자에 지정된 블록 장치의 파일시스템 마운트

- 커널 명령 줄 인자 (Documentation/kernel-parameters.txt)

  debug
  init=
  panic

  quiet
  rdinit=
  ro / rw

  root=
  rootdelay= / rootwait= mmc 인 경우 설정 필요
  rootfstype= jffs2인 경우 설정 필요

05 루트 파일시스템 구성

- 최소한의 루트 파일시스템 구성요소

init
shell(bash)
daemon
공유 라이브러리
설정 파일 (/etc)
장치 노드(/dev)
/proc
/sys
커널 모듈 (/lib/modules/〈커널버전〉)

(교재 184~185 참조)

루트 파일시스템 구성

- busybox 또는 toybox 프로젝트 : 리눅스 필수 기능을 수행하는 도구를 하나의 바이너리로 묶어 제공

```
$ git clone git://busybox.net/busybox.git
$ cd  busybox
$ make distclean
$ make defconfig
$ make menuconfig
$ make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu-
$ make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- CONFIG_PREFIX=./rootfs  install
```

후속 과정 : 교재 194~

06 빌드 시스템

- 임베디드 빌드 시스템 자동화 도구

Buildroot : gnu make와 Kconfig를 이용하는 시스템 (https://buildroot.org)
루트 파일시스템 이미지 빌드가 주 목적

OpenEmbedded : Yocto를 비롯한 빌드 시스템의 코어 컴포넌트

Yocto : 메타데이터, 툴, 문서와 함께 OpenEmdedded의 코어 부분을 확장한 시스템,
가장 인기있는 빌드 시스템 (https://yoctoproject.org)
타깃 시스템 정의를 통해 복잡한 임베디드 장치를 적절히 빌드 할 수 있음
효과적으로 커스터마이징한 리눅스 배포판 생성

Yocto 프로젝트

- 툴체인 + 부트로더 + 커널 + 루트 파일시스템 빌드
- 레시피 그룹(파이썬, 쉘 스크립트)을 중심으로 구성
- BitBake 태스크 스케줄러에 의해 레시피에 설정된 사항을 생성
- 주요 구성요소

구성요소	수행 내용
OE-Core	OpenEmbedded 와 공유하는 코어 메타데이터
BitBake	태스크 스케줄러
Poky	레퍼런스 배포판
Documentation	사용자 매뉴얼, 컴포넌트 개발 지침서
Toaster	BitBake와 그 메타데이터를 위한 웹 기반 인터페이스

Yocto 프로젝트

- Yocto 프로젝트 퀵 빌드 가이드
  (https://docs.yoctoproject.org/current/brief-yoctoprojectqs/index.html)

- 필요한 패키지 설치

```
$ 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 python3-subunit zstd liblz4-tool file locales libacl1
# 언어 환경 설정
$ sudo locale-gen en_US.UTF-8
```

Yocto 설치

- poky 시스템 다운로드

```
$ git clone –b kirkstone git://git.yoctoproject.org/poky.git
$ cd poky
```

- 환경 설정을 위한 스크립트 수행

```
$ source oe-init-build-env  [build-qemuarm]      # 빌드 워킹 디렉토리 생성 후 이동
```

- conf/local.conf : 빌드하려는 장치의 스펙과 빌드 환경, MACHINE 환경변수 설정!
- conf/bblayer.conf : 사용하려는 메타레이어의 경로 설정

Yocto 빌드

- bitbake 의 target (루트 파일시스템 이미지)

```
core-image-minimal : 테스트용, 커스텀 이미지의 기초가 되는 콘솔 기반의 소형 시스템
core-image-full-cmdline : 콘솔 기반 시스템으로 표준 CLI 환경과 타깃 장치에 대한 완전한 지원 시스템
core-image-sato : gnome 데스크탑 시스템
core-image-weston : wayland 를 지원하는 임베디드 그래픽 시스템
meta-toolchain : cross-toolchain 빌드 시스템
meta-ide-support : 통합 개발환경 도구 빌드 시스템
```

- bitbake를 이용한 빌드

```
$ bitbake core-image-minimal
```

약 40GB 저장 공간 필요

Yocto

- bitbake 완료 후 tmp/ 디렉토리

work/	빌드 디렉토리와 루트파일시스템을 위한 작업 영역
deploy/	타깃에 배포할 최종 바이너리
deploy/images/[machine name]/	타깃에서 실행될 부트로더, 커널, 루트파일시스템 이미지
deploy/rpm/	이미지 구성했던 SW 패키지
deploy/licenses/	각 패키지에서 추출한 라이선스 파일

Yocto

- QEMU 타깃 실행 (source oe-init-build-env 적용한 터미널에서 수행)

```
$ runqemu qemuarm nographic
```

종료 : ctrl+A+x

Yocto

- 메타데이터 레이어

meta	OpenEmbedded 코어 및 poky 에 대한 일부 메타데이터
meta-poky	Poky 배포판에 대한 메타데이터
meta-yocto-bsp	Yocto를 지원하는 시스템의 BSP
meta-qt5	Qt5 라이브러리와 유틸리티
meta-intel	Intel CPU와 SoC용 BSP
meta-raspberrypi	라즈베리파이 보드용 BSP
meta-ti	TI ARM 기반 SoC용 BSP

레이어 추가 : 〈빌드디렉토리〉/conf/bblayer.conf 파일
유용한 레이어 목록 : https://layers.openembedded.org/layerindex)

Yocto

- 기본 레시피 + 메타데이터 레이어를 추가하는 방식으로 확장
- 각 레이어간의 의존성 및 호환되는 yocto 프로젝트 버전 확인 필요(README)
- 각 레이어는 최소한 하나 이상의 conf/local.conf와 README 파일 및 라이선스로 구성

Yocto

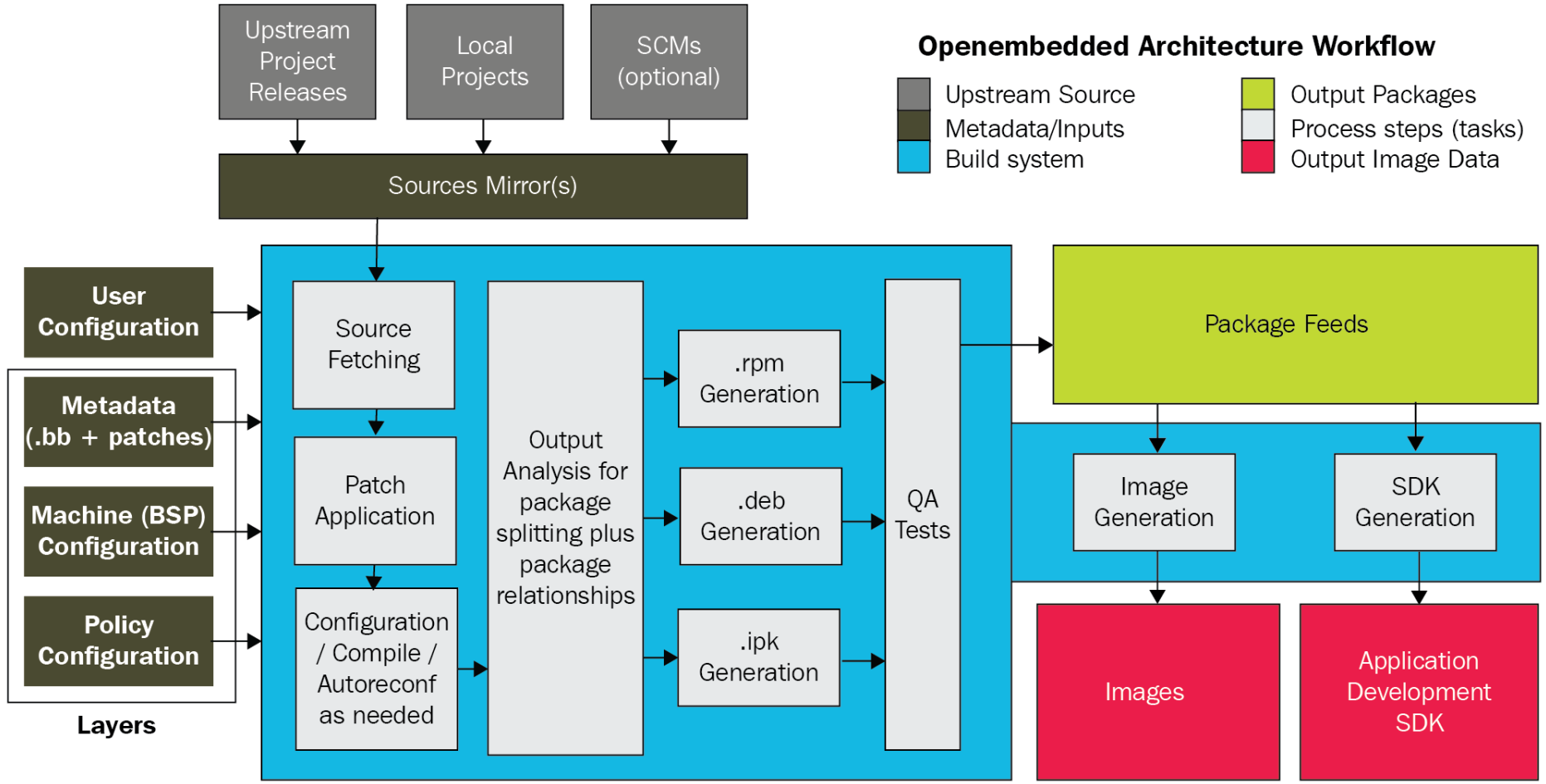
- 라즈베리파이 용 레이어 다운로드

```
$ git clone –b kirkstone git://git.openembedded.org/meta-openembedded
$ git clone –b kirkstone git://git.yoctoproject.org/meta-raspberrypi
```

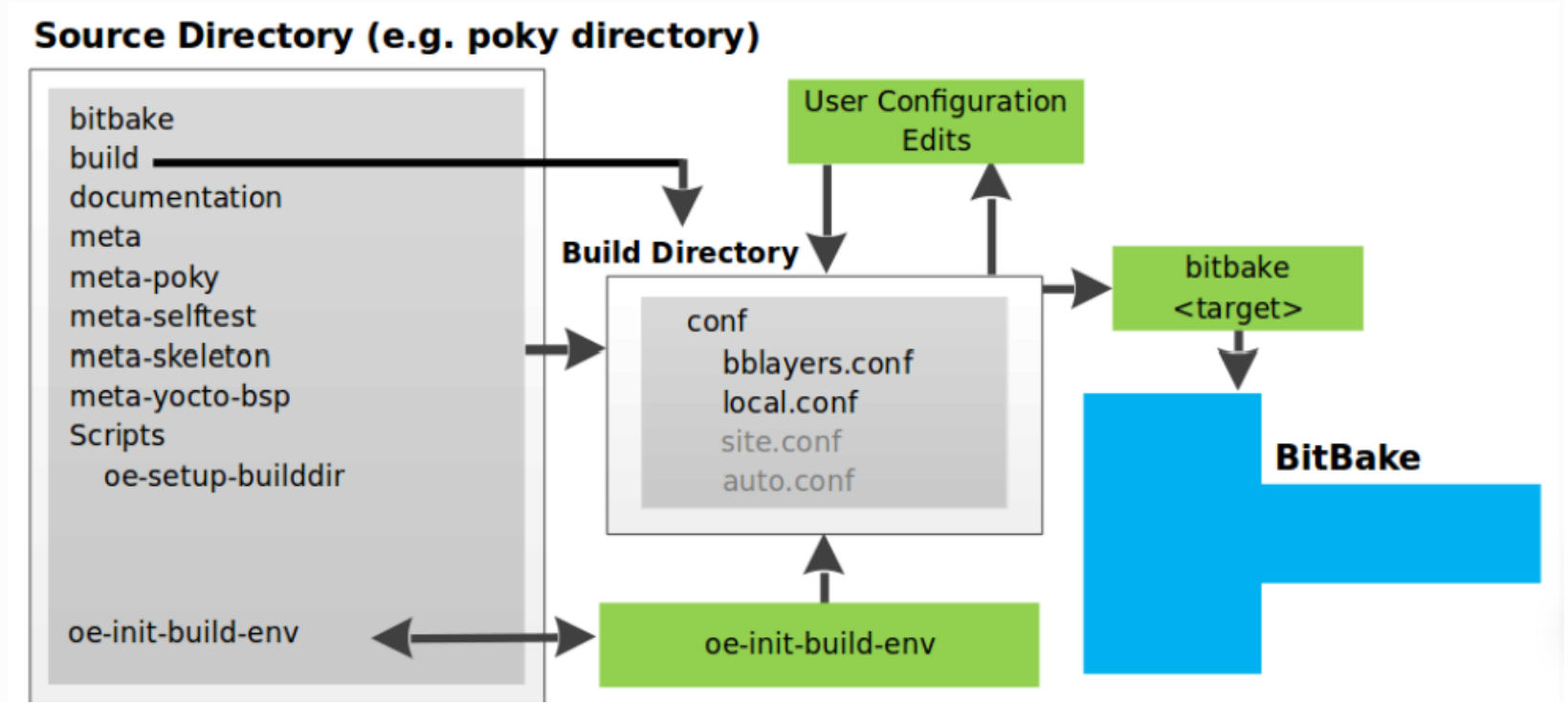
- 환경 변수 설정

```
$ source oe-init-build-env build-rpi5
$ vim conf/local.conf
→ MACHINE ??= “raspberrypi5”    수정

$ vim conf/bblayers.conf
→ BBLAYERS ?= “ \
    …
  /home/ubuntu/pi_bsp/yocto/poky/meta-raspberrypi \    추가
  “
```


Openembedded Architecture Workflow
Upstream Source
Metadata/Inputs
Build system
Output Packages
Process steps (tasks)
Output Image Data
Upstream Project Releases
Local Projects
SCMs (optional)
Sources Mirror(s)
User Configuration
Metadata (.bb + patches)
Machine (BSP) Configuration
Policy Configuration
Layers
Source Fetching
Patch Application
Configuration / Compile / Autoreconf as needed
Output Analysis for package splitting plus package relationships
.rpm Generation
.deb Generation
.ipk Generation
QA Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK

Yocto(poky) 구성요소



https://docs.yoctoproject.org/overview-manual/concepts.html#recipes

bitbake와 메타 데이터

Recipes	.bb 로 끝나는 파일 소스코드 사본이나 다른 콤포넌트의 의존성 파일을 얻는 방법, 빌드 및 설치 방법을 포함 소프트웨어 빌드에 대한 정보로 구성 파이썬과 쉘 프로그램으로 이루어진 태스크들의 모음
Append	.bbapend로 끝나는 파일 레시피의 세부 항목을 덮어 씌우거나 확장
Include	.inc 로 끝나는 파일 여러 레시피를 위한 공통된 정보 포함, Include 또는 require 키워드를 이용하여 포함
Classes	.bbclass 로 끝나는 파일 공통 빌드 정보 포함, inherit 키워드를 이용하여 상속됨
Configuration	.conf 로 끝나는 파일 프로젝트 빌드 프로세스를 통제하는 구성 변수로 이루어짐

bitbake 명령

https://docs.yoctoproject.org/bitbake/dev/index.html
https://elinux.org/Bitbake_Cheat_Sheet

- 태스크 리스트 확인

```
$ bitbake  -c listtasks  core-image-minimal
```

〈주요 task〉
do_fetch : source code를 fetch 작업 수행
do_unpack: source code의 압축 해제 작업 수행 (tar.gz, zip, xz, tar ..)
do_patch: source code에 적용할 patch가 있는 경우 적용하는 작업 수행
do_configure: source code에 configure script가 있을 경우 이를 수행하는 작업
do_compile: compile 작업 수행
do_install: build 결과물을 rootfs에 포함시키는 작업(설치)
do_package: 패키지 생성 작업
do_rootfs: rootfs 이미지 생성 작업 수행

bitbake 명령 사용

- 소스 코드 다운로드

```
$ bitbake   -c fetch  타깃명
$ bitbake  core-image-minimal  --runall=fetch
```

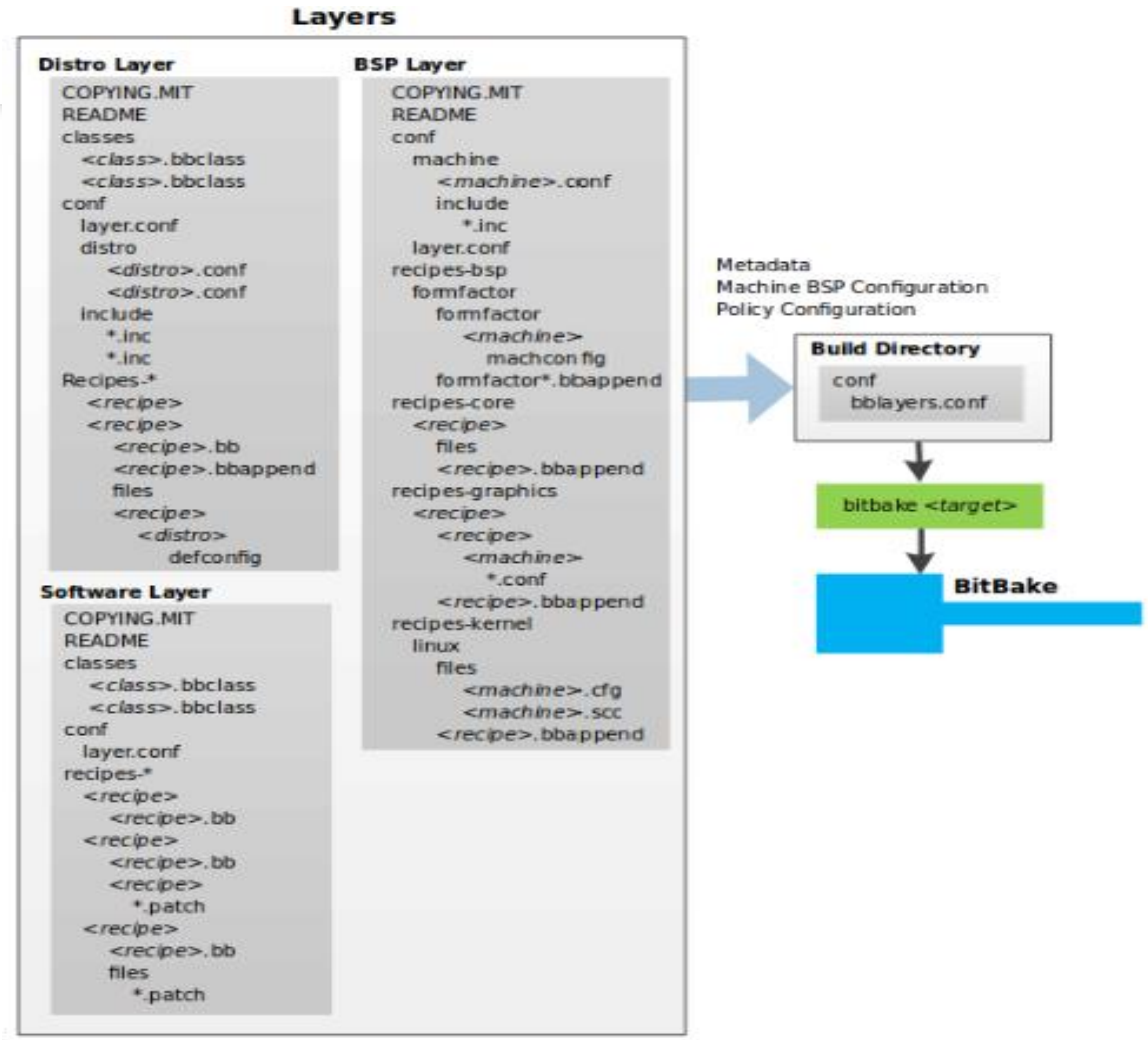
- 환경 변수 확인

```
bitbake recipename -e | grep VARIABLENAME
```

https://docs.yoctoproject.org/bitbake/dev/bitbake-user-manual/bitbake-user-manual-ref-variables-context.html

yocto 를 이용한 프로그램 개발

- layer 생성
- 소스파일 작성
- recipes 생성
- image 에 추가



bitbake-layers 명령

- 레이어 생성/확인/추가/삭제

```
$ bitbake-layers create-layer  layer명
$ bitbake-layers  show-layers
$ bitbake-layers add-layer layer명
$ bitbake-layers remove-layer layer명
```

소스 코드 작성

- 교재 263 페이지 참조하여 레이어 디렉토리에 디렉토리 생성 및 소스 파일 생성

```
 ~/poky$ tree meta-hello
meta-hello
├── COPYING.MIT
├── README
├── conf
│   └── layer.conf
├── recipes-example
│   └── example
│       └── example_0.1.bb
└── recipes-hello
    └── hello
        ├── files
        │   └── hello.c
        └── hello_1.0.bb
```

recipes 작성

- 레시피 파일명 : *〈패키지명〉_〈버전〉*.bb

SUMMARY : 레시피에 대한 간략한 설명
LICENSE : 라이선스 종류(MIT, GPL, BSD etc.)
LIC_FILES_CHKSUM : 라이선스 파일 위치 및 이 파일의 md5 checksum(md5sum 도구)
SRC_URI : 소스 파일

do_compile : 컴파일 태스크 작성
do_install : 최종 실행 파일 위치 설치

교재 263 페이지 참조
hello_1.0.bb 생성

```
S = "${WORKDIR}/sources"
UNPACKDIR = "${S}"
```

```
FILES${PN} += "${bindir}/hello"
```

image에 추가

- *⟨BuildDirectory⟩*/conf/local.conf 에 추가

```
IMAGE_INSTALL_append = "hello"
```

OR

```
IMAGE_INSTALL += "hello"
```

OR

```
CORE_IMAGE_EXTRA_INSTALL += "hello"
```

- bitbake 로 이미지 다시 생성
- runqemu 로 확인

yocto – SDK 생성

- 툴 체인 생성

```
$ bitbake -c populate_sdk core-image-minimal
```

- 기본 툴체인만 설치 (C/C++ 크로스컴파일러, C 라이브러리, 헤더파일)

```
$ bitbake  meta-toolchain
```

- 설치 스크립트 실행 및 확인

```
$ tmp/deploy/sdk/poky-#######.sh

$ source /opt/poky/###/environment-setup-#######
$ $CC 소스파일명 -o 실행파일명
```

yocto – runqemu

- runqemu 도구의 사용

  https://docs.yoctoproject.org/dev/dev-manual/qemu.html

- ssh 원격접속 방법

yocto 관련 사이트

명령어와 사용법 총정리
https://ahyuo79.blogspot.com/2020/02/yocto-recipe.html

개념정리
https://slowbootkernelhacks.blogspot.com/2016/12/yocto-project.html

사용 예제 정리
https://github.com/Munawar-git/YoctoTutorials/tree/master/00_Yocto_Intro