Toradex
Linux OS
カスタムマニュアル
BSP5用

本マニュアルは岡本無線電機株式会社が独自作成したものでありメーカーが保証した内容ではありません。万が一本マニュアルに間違いがあり、事故が生じたとしても岡本無線電機株式会社は一切の責任を問われないものとさせていただきます。

変更履歴

バージョン	更新日付	変更内容
1.00	2022/11/07	BSP5用に作成

本マニュアルについて

本マニュアルではOpen Embedded**の使い方やレシピの読み方、トラデックスの**CPU**ボードに搭載する**rootfs**や**Linux OS,**ブートローダーなどをカスタマイズする手順を記述しています。**

参考：
http://www.openembedded.org/wiki/Main_Page
https://docs.yoctoproject.org/dev-manual/index.html
https://developer.toradex.com/knowledge-base/splash-screen-linux
https://developer.toradex.com/knowledge-base/display-output-resolution-and-timings-linux

1．実行環境

本マニュアルの実行環境は下記です。

仮想化ソフト：VMWARE Player v15.5.7
Host OS： Windows 10 21H2
Guest OS： Ubuntu Desktop 20.04LTS 64bit(**英語版**)
BSP：v5.7
CPU**モジュール**：Verdin iMX8M Plus Quad 4GB Wi-Fi / Bluetooth IT V1.1A
キャリアボード：Verdin**開発ボード** Rev 1.1C + **アクセサリーキット**
ディスプレイ：10.1**インチディスプレイ**1.0A **静電式タッチパネル付き** + DSI to LVDS**アダプタ** 1.1

2．事前準備

本マニュアルはLinux OS**イメージ開発マニュアルの内容をすべて終えた状態で進めています。**

3．前提知識

Linux OS**イメージ開発マニュアルの内容、**TEZI**による**OS**書き込みをご理解いただいた状態を前提としています。**

4. 注意点

オープンソース系を利用した開発に共通することですがOpen Embeddedをすべてを理解しようとするときりがなく開発効率を損ないます。必要なタイミングで必要な知識を身につけるというスタンスで理解することを推奨いたします。本マニュアルで扱うことはOpen Embeddedによる開発でよく使う内容に絞っています。

開発環境(パソコン)と実行環境(モジュール)の違いをわかりやすくするためにコマンドの表記の前に下記をつけています。
開発環境(パソコン)上で入力するコマンド：[Ubuntu]$
実行環境(モジュール)上で入力するコマンド(Linux) ：[Module]#
実行環境(モジュール)上で入力するコマンド(U-Boot)：[U-Boot]#

コピーについて
本マニュアル内のコマンドなどをコピーした場合、改行が入ったり「-」が抜けてしまうことがあるのでご注意ください。一度テキストエディタなどに張り付けてコピーした内容をご確認ください。

Open Embeddedの大雑把な仕組み

Open Embeddedはbitbakeというコマンドを実行することでOSイメージやミドルウェアなどをビルドしRootFSやKernel、ブートローダーなどを作成するビルドシステムです。Open Embedded自体はPythonで作成されています。
bitbakeのビルド処理は大雑把に下記のような手順が実行されます。

do_fetch: ソースコードをダウンロード
do_unpack: ソースコードを解凍(展開)
do_patch: パッチを適用
do_populate_lic: ライセンスファイルのコピー
do_prepare_recipe_sysroot: makeに必要なツールの準備
do_configure: configureを実行
do_compile: makeを実行
do_install: make install を実行
do_package: パッケージをサブセットに分離など
do_populate_sysroot: sysroot にインストール
do_packagedata: パッケージデータの移動
do_package_qa: パッケージのチェック処理
do_package_write_ipk: パッケージの作成 (ipk)
do_rm_work:一時ファイルの削除

最もシンプルなbitbakeの使用コマンドは下記のようにbitbakeするターゲット名を指定するだけです。
bitabke <ターゲット名(レシピ名)>
このコマンドを行うだけでソースコードのダウンロードからパッチ適用、makeなどを行いRootFSやカーネルがすべて自動で出力されます。このビルドターゲットがどのようなビルドの内容なのかという定義を「レシピ」といわれるファイルで定義します。
レシピにはソースコードのダウンロードURIやパッチファイルの指定、各工程前後のシェルスクリプトの実行などを定義します。
レシピにはソースコードなどからひとつの生成物を作成するものとソースコードはなく複数のレシピのみで構成されるものがあります。例えばミドルウェアなどは前者であり、OSイメージなどは元のソースコードがあるわけではなくミドルウェアやカーネルの組み合わせでしかないため後者になります。レシピの書き方はシェルに似ていますが独自定義などもありよく使用する定義値などは覚える必要があります。Open EmbeddedはRootFSやKernelなどの最終的な出力物以外にもターゲット向けコンパイラやautoconf、unzipなど出力物を生成するために必要なツール類も自動で出力します。

ベースのOSイメージのレシピを選択

トラデックスが用意したOSイメージのレシピは/work/oe-core/layers/meta-toradex-demos/recipes-images/images/配下にあります。BSP5.Xにおいてはtdx-reference-minimal-image.bb、tdx-reference-multimedia-imageのみ対応しています。ほかにもファイルは存在しますがサポートはしていません。
下記のようにbitbakeコマンドでOSイメージを作成することができます。

[Ubuntu]$ bitbake tdx-reference-multimedia-image

bitbakeの一部のコマンドだけを実行

下記コマンドでbitbake**の一部の処理だけを実行できます。**
[Ubuntu]$ bitbake <**レシピ名**> -c <**処理名**>

例えば下記のように入力することでカーネルのソースコードのダウンロードだけを行うことができます。
[Ubuntu]$ bitbake virtual/kernel -c fetch

各レシピが持つ処理一覧は下記コマンドで表示可能です。
[Ubuntu]$ bitbake <**レシピ名**> -c listtasks

レシピ内の演算子

レシピ内ではいくつかの演算子があります。意味は下記のようになっています。

= 通常の設定
?= 未定義の場合に設定
??= ?=よりさらに弱い設定
:= 即時代入
+= 後ろへ追加挿入(スペースあり)
=+ 前へ追加挿入(スペースあり)
.= 後ろへ追加挿入(スペースなし)
=. 前へ追加挿入(スペースなし)
_append 解析後後ろへ追加挿入(スペースなし)
_prepend 解析後前へ追加挿入(スペースなし)

レシピ内の定義の読み方

下記はよく使用する定義です。

SRC_URI：ソースコードやパッチなどの入手先
SRCBRANCH：ブランチ名
SRCREV：コミット番号
IMAGE_INSTALL：インストールを行うパッケージ
PV：バージョン
PR：リビジョン
DEPENDS：ビルド時に依存するパッケージ
RDEPENDS：実行時に依存するパッケージ
COMPATIBLE_MACHINE：互換性のあるターゲット
<定義>_append_<ターゲット名>：該当ターゲット時のみ適用される定義
MACHINE：ターゲット名
S:ソースディレクトリ
B：ビルドディレクトリ
D：インストール先ディレクトリ

DISTRO_FEATURES：ディストリビューションに対しての機能性を定義します。
各レシピ内でこの定義に含まれるものを参照してビルドを行います。
例えばDISTRO_FEATURESにx11が含まれていればx11に関連するコンパイルオプションや設定が取り込まれます。
x11を使用しない場合はDISTRO_FEATUREからx11を取り除くことでその分OSイメージが小さくなります。

MACHINE_FEATURES：サポートするハードウェアを定義します。
各レシピ内でこの定義に含まれるものを参照してビルドを行います。
例えばMACHINE_FEATURESにbluetooth が含まれていればbluetoothに関連する機能や設定が取り込まれます。
bluetoothを使用しない場合はMACHINE_FEATURESからbluetoothを取り除くことでその分OSイメージが小さくなります。

レシピ内の処理の読み方

レシピ内の処理には下記のようなdo_<処理名>に対してそれぞれ事前処理、処理の上書きが存在します。

do_fetch: ソースコードをダウンロード
do_unpack: ソースコードを解凍(展開)
do_patch: パッチを適用
do_configure: configureを実行
do_compile: makeを実行
do_install: make install を実行
do_populate_sysroot: sysroot にインストール
do_package: パッケージ化用のディレクトリにインストール
do_package_write: パッケージの作成 (ipk, deb, rpm)
do_build: ビルド終了用のタスク
do_rm_work:一時ファイルの削除

例えばdo_configure_prependという処理を記述すればconfigureの以前処理を行うスクリプトを記述します。
do_installという処理を記述すればinstallの処理を上書きします。

自作タスクの登録
addtaskで自作したタスクの登録が可能です。
addtask <新規に定期したタスク名> after <タスク名> before <タスク名>

例えば下記の記述はunpackとpatchの間でhelloを出力するタスクを登録しています。

```
do_echo () {
    echo "hello"
}
addtask do_echo after do_unpack before do_patch
```

レシピの実行ログと結果のログ

bitbakeを実行後どのような処理が実行されたのか下記に実行コマンドのログが出力されます。
/work/oe-core/build/tmp/work/<アーキテクチャ名>/<レシピ名>/<バージョン>/temp/run.do_<処理名>

例えばVerdin-iMX8MPのカーネルのconfigureのログの場合下記のようになります。(バージョンによりパスは異なります。)
/work/oe-core/build/tmp/work/verdin_imx8mp-tdx-linux/linux-toradex/5.4.193+gitAUTOINC+f782992971-r0/temp/run.do_configure

同ディレクトリにlog.do_<処理名>というファイル名で実行結果のログが出力されます。

例えばVerdin-iMX8MPのカーネルのcompileのログの場合下記のようになります。
/work/oe-core/build/tmp/work/verdin_imx8mp-tdx-linux/linux-toradex/5.4.193+gitAUTOINC+f782992971-r0/temp/log.do_compile

log.task_orderに実行したタスクのオーダーと各タスクに対するログのファイル名が出力されます。

レシピの追加/削除

例えばzipコマンドを必要とするOS イメージを開発する場合、OSイメージのレシピ(tdx-reference-multimedia-image.bb)をベースとして下記のようにレシピ内のIMAGE_INSTALLに項目を追加しzipを追加します。

```
IMAGE_INSTALL += " ¥
    ${@bb.utils.contains('DISTRO_FEATURES', 'wayland', ¥
                    'weston weston-init weston-examples ${APP_LAUNCH_WAYLAND}', '', d)} ¥
    ${@bb.utils.contains('DISTRO_FEATURES', 'x11 wayland', ¥
                    'weston-xwayland xterm', ¥
       bb.utils.contains('DISTRO_FEATURES', 'x11', '${APP_LAUNCH_X11}', '', d), d)} ¥
    ¥
    .
    .
    .

    gpicview ¥
    media-files ¥
    zip ¥
"
```

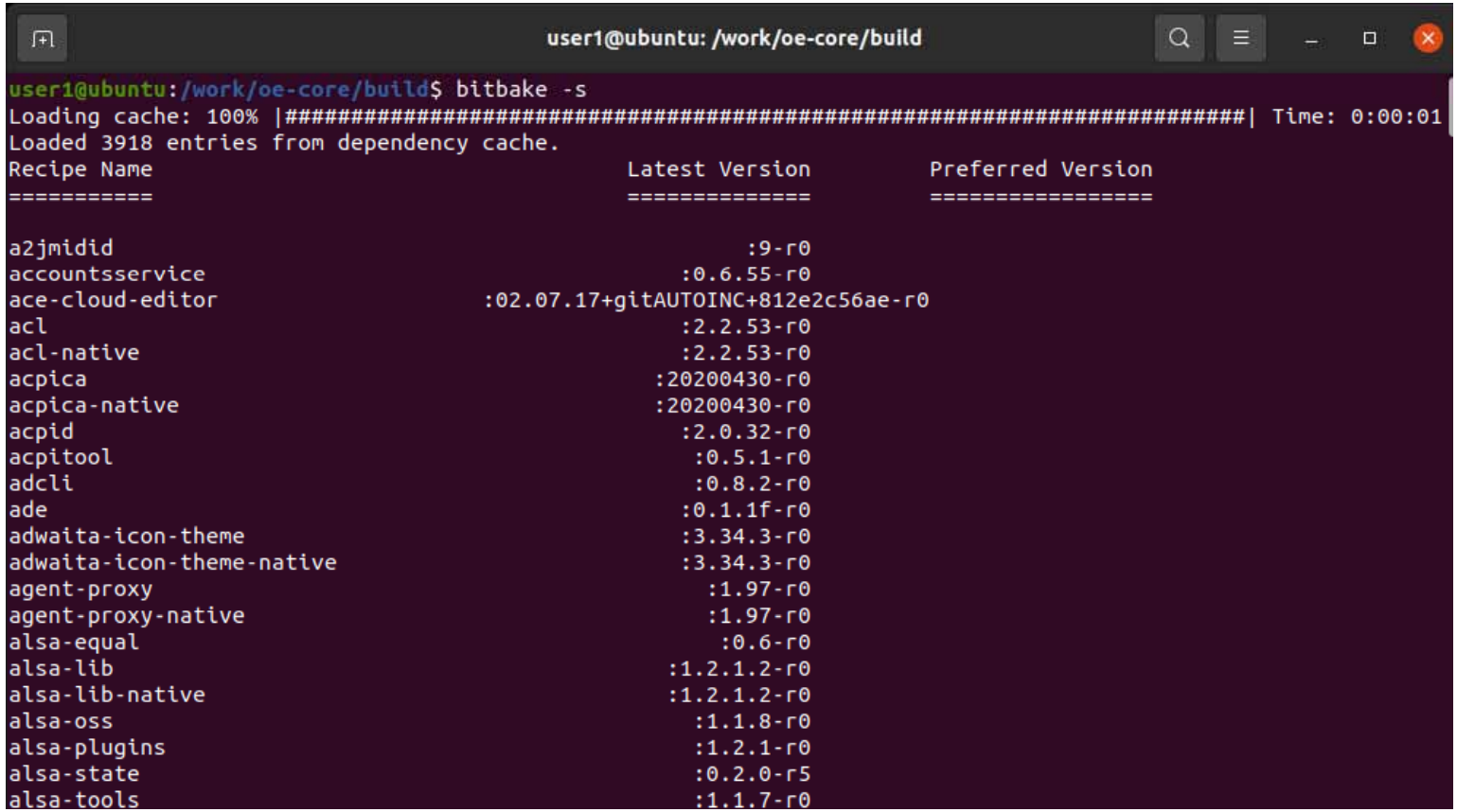
変更後、bitbakeを行うとzipコマンドが追加されたOSイメージが作成されます。
出力されたOSイメージを書き込み、動作させることでこれらの機能が追加されたことが確認できます。
逆に不要なものがあればレシピを削除することで機能を削除することができます。

local.confに追加することも可能です。こちらのほうがもともとのレシピを変更せずに対応可能です。
(zipの手前はスペースが必要です。)
IMAGE_INSTALL_append = " zip"

削除する場合は下記のような記述をします。 gpicviewを削除する例
IMAGE_INSTALL_remove = "gpicview"

存在するレシピの確認

OSイメージのレシピに機能を追加するにはどのような機能が存在するかを把握しておく必要があります。
下記のように入力するとBSP内（レシピ全体）に含まれるパッケージとバージョンを表示できます。
[Ubuntu]$ bitbake -s

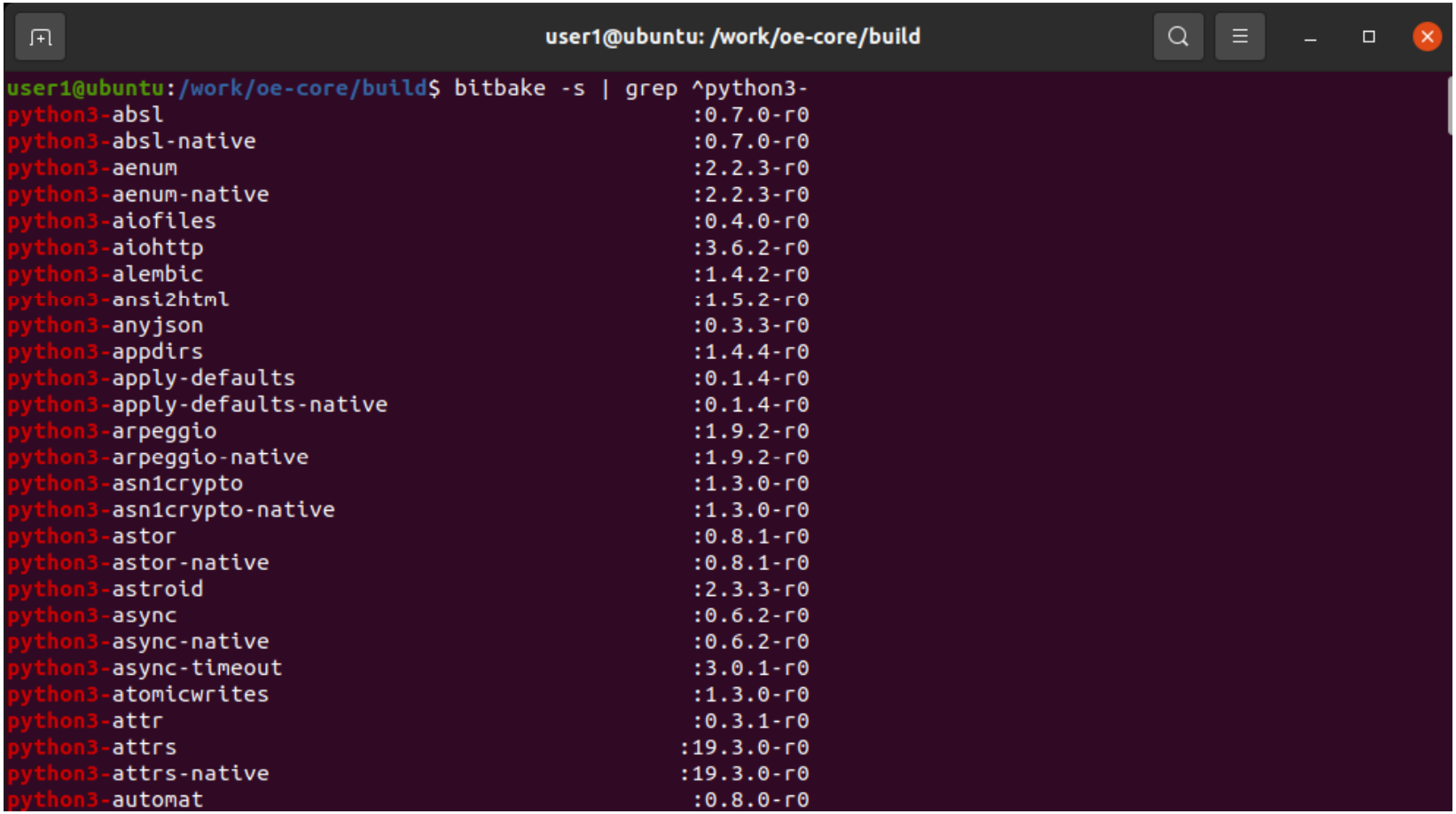


機能ごとに分割されたレシピの追加

例えばOSイメージにpythonを追加する場合、pythonの機能ごとにレシピが分かれていてレシピにpythonだけを追加してもpythonのすべての機能を使用することはできません。消費するリソースを最小限にできるように必要な機能だけをレシピに組み込めるようになっています。

下記コマンドでpython(3系)のレシピ一覧を表示できます。

bitbake -s | grep ^python3-



出力ファイルの追加

レシピから出力されるファイルはレシピを組み込んだだけではすべてのファイルが組み込まれるとは限りません。一部のファイルはレシピ以外にもファイルの出力を指定しないといけないものがあります。

例えば下記のdhcpのレシピはlocal.confなどにIMAGE_INSTALL_append = " dhcp"を追記してもbitbakeはエラーになります。

/work/oe-core/layers/openembedded-core/meta/recipes-connectivity/dhcp/dhcp_4.4.2.bb

レシピ名を指定した場合に出力されるファイルはFILES_${PN}になりますがdhcpのレシピにはFILES_${PN}の指定がありません。

その代わりFILES_${PN}-serverやFILES_${PN}-client、FILES_${PN}-server-configなどが存在します。

dhcpサーバーの機能がほしい場合dhcp-server、クライアントの場合dhcp-client、サーバーの設定ファイルの場合、dhcp-server-configを指定すれば該当機能の出力ファイルがイメージに追加されます。

コンパイルオプションの変更

レシピにはコンパイルオプションがあるものがあります。デフォルトのコンパイルオプションから変更したい場合はlocal.confに下記のように記述します。

PACKAGECONFIG_append_pn-<レシピ名> = “<コンパイルオプション>“

例えばwgetにsslのコンパイルオプションをつけたい場合は下記のように記述します。
PACKAGECONFIG_append_pn-wget = " ssl "

wgetのレシピ内にPACKAGECONFIGの記述があります。
/work/oe-core/layers/openembedded-core/meta/recipes-extended/wget/wget.inc
PACKAGECONFIG[openssl] = "--with-ssl=openssl,,openssl"

上記の記述の意味は下記です。
PACKAGECONFIG[p0] = “p1,p2,p3,p4”
p0の指定があればp1をコンパイルオプションに適用
p0 の指定がなければp2をコンパイルオプションに適用
p0 の指定があればp3をDEPENDSに追加(上記の場合p3は省略されています。)
p0 の指定があればp4をRDEPENDSに追加

各レシピがどのようなコンパイルオプションがあるかはレシピの内容を見る必要があります。
レシピが扱うミドルウェアが持つレシピ内にはないコンパイルオプションを記述したい場合はレシピを変更する必要があります。

依存関係の確認

bitbake -g <レシピ名>で依存関係情報を出力できます。
例えばtdx-reference-multimedia-imageの依存関係情報を出力するには下記のコマンドになります。
[Ubuntu]$ bitbake -g tdx-reference-multimedia-image

コマンドを実行すると下記2つのファイルが出力されます。
pn-buildlist ：指定レシピに含まれるレシピ一覧
task-depends.dot ：各タスクの依存関係一覧

環境情報の確認

bitbake -e <レシピ名>で環境情報を出力できます。レシピ名がない場合はグローバルの環境情報を出力します。
例えばtdx-reference-multimedia-imageの環境情報を出力するには下記のコマンドになります。
[Ubuntu]$ bitbake -e tdx-reference-multimedia-image > log

この情報を見ると内部でDISTRO_FEATURESやMACHINE_FEATURES、IMAGE_INSTALL、RDEPENDSなどにどんな値が設定されているかを確認することができます。

仮想レシピ名

いくつかの仮想レシピ名が存在します。
これらは実体のレシピ名を知らなくても使える便利なものです。

virtual/kernel
linux**カーネル本体を示す名前　実体は**linux-toradex

virtual/bootloader
ブートローダーを示す名前　実体はu-boot-toradex

レシピのファイルの所在を調べる

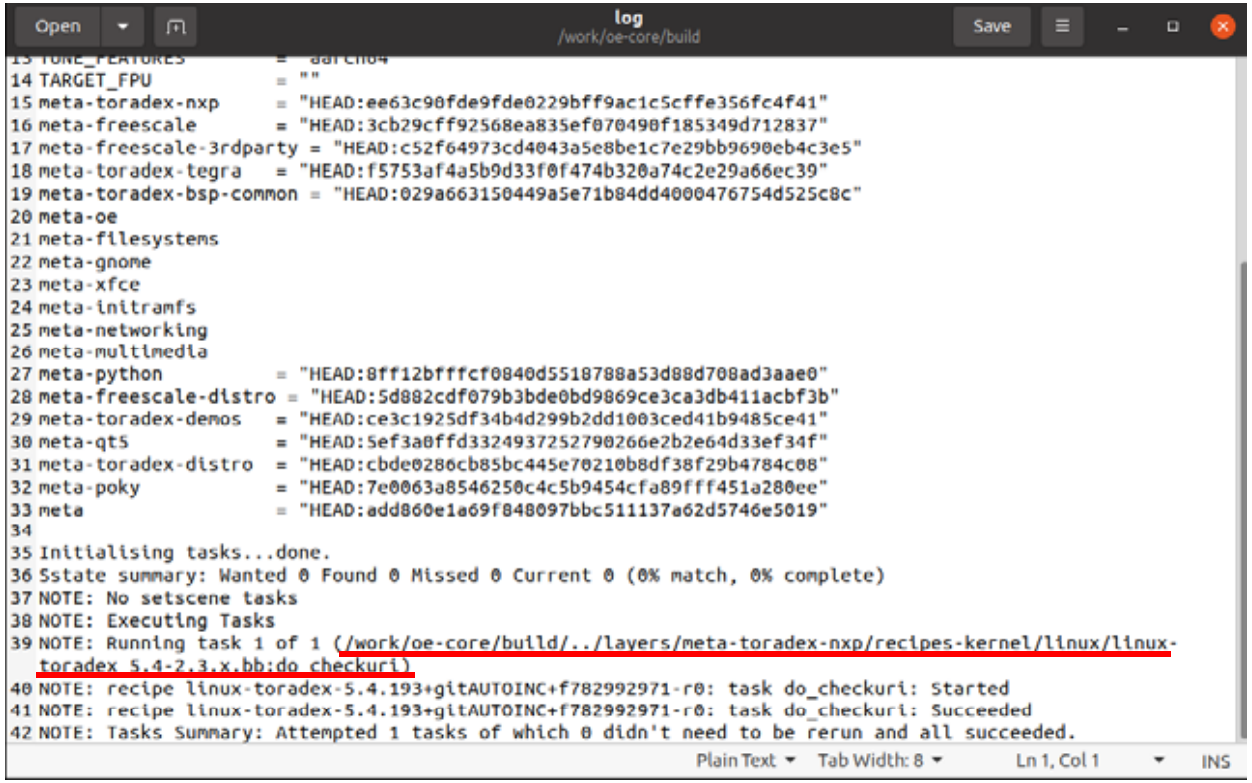
カーネルのレシピ名はすべてのモジュール共通でlinux-toradexです。このレシピのファイルがどこに存在するかを下記コマンドで検索するといくつか候補が出てきますがどれに当たるのかはわかりません。(同名レシピのバージョン違いがいくつか存在します。)

[Ubuntu]$ find /work/oe-core/layers/ -name "linux-toradex*"

確実にファイル名を知るには下記のコマンドで確認することができます。

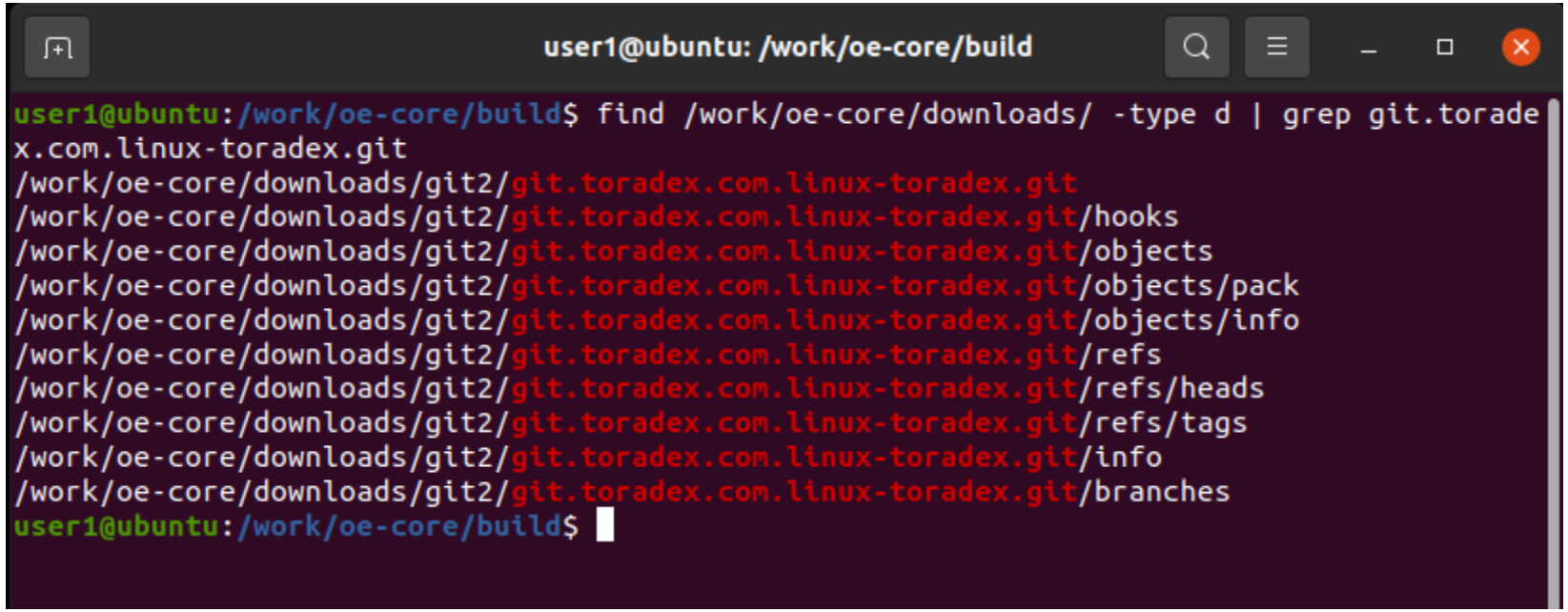
[Ubuntu]$ bitbake -c checkuri linux-toradex > log

ログの最後のほうにlinux-toradex_XXX.bbというファイル名が出力されているのがレシピのファイル名です。



レシピのダウンロードファイルを調べる

bitbakeでダウンロードしたものはデフォルト設定ではoe-core/build/downloadsに格納されます。
レシピのSRC_URIがhttpやftpの場合ファイルで保存されるためファイル名でわかります。
gitでダウンロードした場合はgitのディレクトリが作成されます。どこに保存されるかを調べるにはgitのURLからわかります。
例えば下記のLinuxカーネルのレシピの場合
/work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/linux-toradex_5.4-2.3.x.bb
SRC_URIはgit://git.toradex.com/linux-toradex.git;protocol=https;branch=${SRCBRANCH};name=machineになっています。
git://以下の部分を抽出し/は.に置き換えられてダウンロードしたディレクトリはgit.toradex.com.linux-toradex.gitという名前になります。
下記のようなコマンドを入力すると所在がわかります。
find /work/oe-core/downloads/ -type d | grep git.toradex.com.linux-toradex.git
下記の場合、/work/oe-core/downloads/git2/git.toradex.com.linux-toradex.git/がgitのディレクトリになります。



クリーン

一部レシピやパッチを作成してソースコードを変更した場合にはクリーンしないとビルドしてくれません。
例えばkernelを変更した場合はコマンド実行しクリーンしてからビルドを行います。
bitbake -c cleansstate virtual/kernel && bitbake virtual/kernel

何かおかしくなってすべてをビルドしなおしたい場合はすべてをクリーンする必要がありますが非常に手間がかかります。
その場合はbitbakeで出力されたファイルやディレクトリをすべて削除してから再度bitbakeを行います。
対象は下記です。ダウンロードしたソースコードとconfディレクトリだけはそのまま残します。
/work/oe-core/sstate-cache
/work/oe-core/build/deploy
/work/oe-core/build/buildhistory
/work/oe-core/build/cache
/work/oe-core/build/tmp

レイヤーの表示

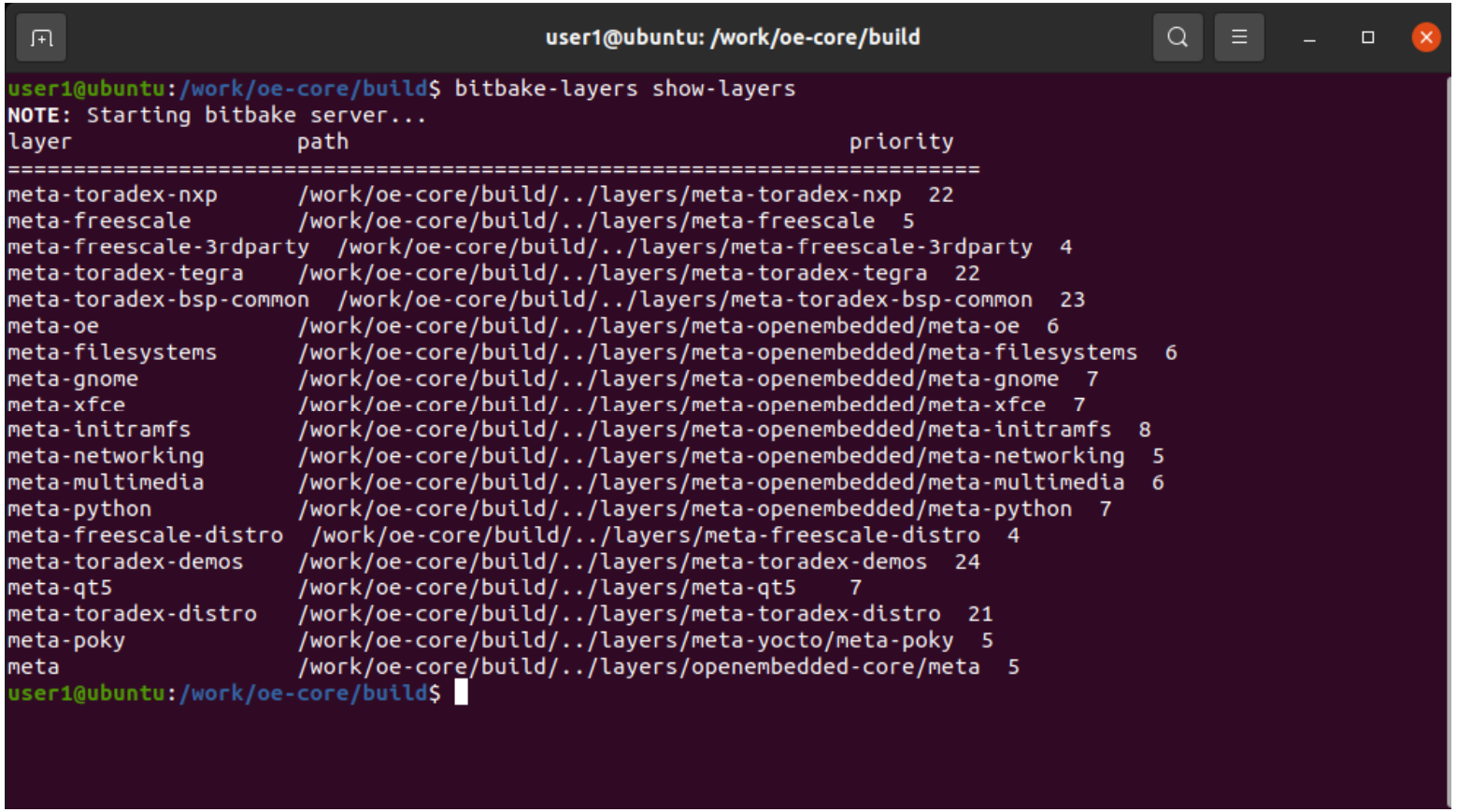
OpenEmbeddedのレシピは複数のレイヤーから構成されています。各々の開発プロジェクトの違いなどによりレイヤーとして存在しています。構成されるレイヤーは下記のコマンドで確認できます。

bitbake-layers show-layers

priorityは処理順序などに影響します。

どのようなレイヤーが有効になっているかは下記に設定があります。

/work/oe-core/build/conf/bblayers.conf



レイヤーの作成

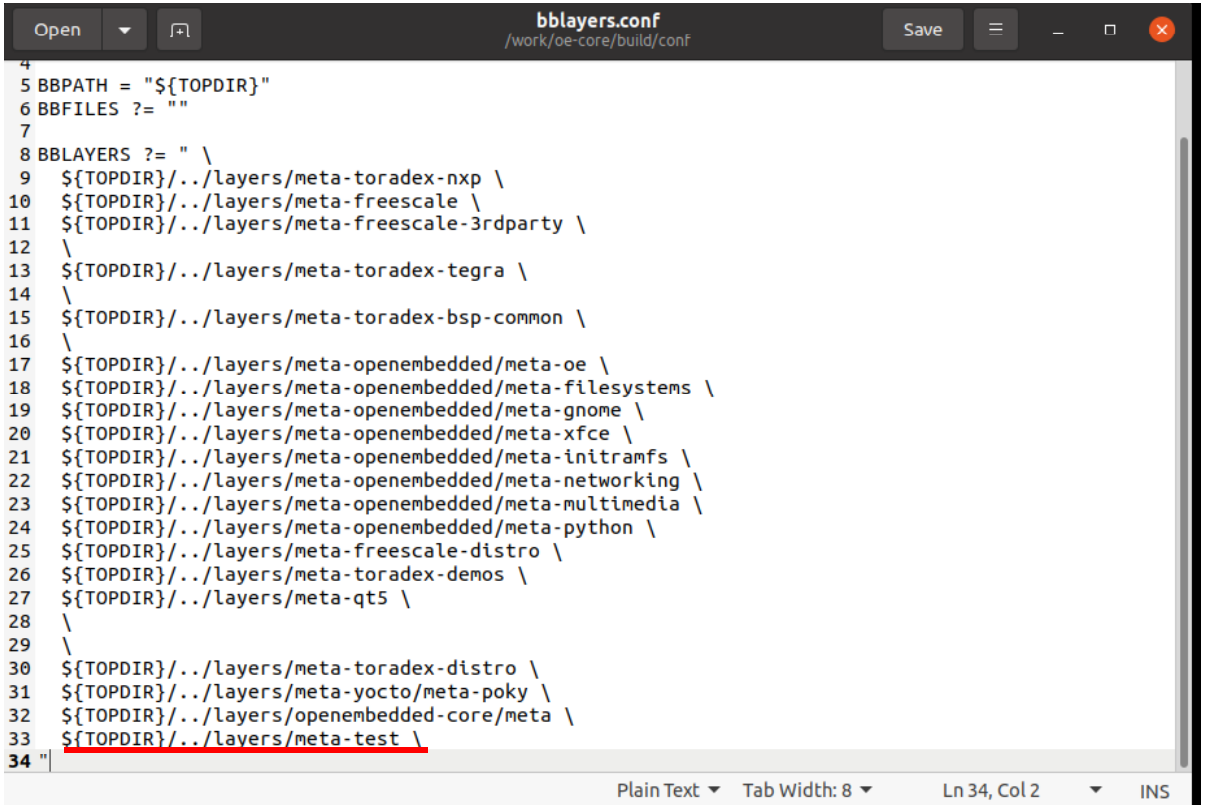
例えばmeta-testという独自のレイヤーを作成する場合には下記のようなコマンドで追加します。

bitbake-layers create-layer ../layers/meta-test

上記コマンドで../layers/meta-testが作成され中にテンプレートのようなものが入っています。

meta-testを有効にするには/work/oe-core/build/conf/bblayers.confに追記する必要があります。

BBLAYERSに設定されているレイヤーは順番に読み込みが行われます。依存関係があるため追加順序に気を使う必要があります。かならず最後に追加してください。



自作レシピの作成

通常はレシピにアプリケーションのソースコードなどを記述しますがアプリケーションの開発は開発効率の都合上、別途IDEなどを用いて開発することが多いためソースコードは追加せず開発完了後のコンパイル済み実行ファイルなどを追加するという手法の方が簡単です。他にはアプリケーションを自動起動するためのsystemd用のサービスや設定ファイルなど主にrootfsに組み込む内容になります。

例としてOS起動後、時刻のログを出力するシェルを自動実行するようなサービスを追加します。

レシピが読み込むファイルを格納するディレクトリ作成

[Ubuntu]$ mkdir ../layers/meta-test/recipes-example/example/files

アプリケーションの作成

[Ubuntu]$ gedit ../layers/meta-test/recipes-example/example/files/app.sh

app.sh**の内容**

- -

#!/bin/sh

date >> /home/root/log

- -

アプリケーション自動実行サービスの作成

[Ubuntu]$ gedit ../layers/meta-test/recipes-example/example/files/app.service

app.service**の内容**

- -

[Unit]
Description = Application

[Service]
ExecStart = /home/root/app.sh
Type = simple

[Install]
WantedBy = multi-user.target

- -

レシピの修正をしてOSイメージを書き込めばシェルが自動起動して起動時刻のログが出力されていることを確認できます。
[Ubuntu]$ gedit ../layers/meta-test/recipes-example/example/app_1.0.0.bb

app_1.0.0.bb**の内容**
- -
```
SUMMARY = "Test recipe"
LICENSE = "CLOSED"
SRC_URI = " ¥
    file://app.sh ¥
    file://app.service ¥
"

inherit systemd

FILES_${PN} = " ¥
    ${ROOT_HOME}/app.sh ¥
    ${sysconfdir}/systemd/system/app.service ¥
"
SYSTEMD_SERVICE_${PN} = "app.service"
SYSTEMD_AUTO_ENABLE_${PN} = "enable"

do_install() {
    install -d ${D}${ROOT_HOME}
    install -m 0755 ${WORKDIR}/app.sh  ${D}${ROOT_HOME}

    install -d ${D}${sysconfdir}/systemd/system/
    install -m 0644 ${WORKDIR}/app.service ${D}${sysconfdir}/systemd/system/
}
```

ユーザーの追加、パスワードの設定

モジュール上でLinuxが動作している状態ではuseraddコマンドやpasswdコマンドでユーザーの追加やパスワードの設定が可能です。ですが量産時にはこれらを手動で行うのは非効率です。

OSイメージに直接ユーザーの追加やパスワードの設定をすることができます。

下記のような設定を行います。

(デフォルト状態ではrootユーザーがパスワードなしで存在します。)

local.confなどに下記を追記します。

```
INHERIT += "extrausers"
#rootのパスワードを設定する例
EXTRA_USERS_PARAMS = "usermod -P password root"

#上記加えてuser1というユーザーをパスワードがpasswordで作成する例
EXTRA_USERS_PARAMS = " usermod -P password root;useradd -P password user1"
```

Alternativeコマンド

Linuxのコマンドには同名の複数のコマンドが存在します。同名コマンドをどのコマンドと紐づけるかの設定が存在します。
デフォルトのOSイメージにはたくさんのコマンドを使用することができるBusyBoxが搭載されています。
例えばlsコマンドはBusyBoxに搭載されていますがそれとは別にcoreutilsというレシピにもlsコマンドが存在します。
デフォルトのOSイメージにはcoreutilsも搭載されてlsコマンドが二つ存在します。
BusyBoxのlsコマンドは機能が省かれた軽いものですがcoreutilsのlsは機能が多く日本語のファイル名も表示することができます。

起動後にどちらのlsが使用されているかはHelpを見れば判断しやすいです。Helpの最後の方にcoreutilsと記載があります。
[Module]# ls --help

coreutilsのlsコマンドが動作します。どちらを使うかはシンボリックシンクで設定されています。

whichコマンドで所在を調べます。
[Module]# which ls
/bin/ls

シンボリックリンクを調べます。
[Module]# ls -l /bin/ls
lrwxrwxrwx 1 root root 17 May 11 06:16 /bin/ls -> /bin/ls.coreutils

この設定は/usr/lib/opkg/alternatives/配下にあります。
lsの場合は/usr/lib/opkg/alternatives/lsになります。ls以外のコマンドの設定も同様に存在します。
[Module]# cat
/bin/ls
/bin/ls.coreutils 100
/bin/busybox.nosuid 50

/bin/lsに対してシンボリックリンクが/bin/ls.coreutilsと/bin/busybox.nosuidがあることを意味します。後ろの数値は優先度で
優先度が高い方が適用されます。

update-alternativeコマンドでこの設定を変更することができます。

例えば下記コマンドでBusyBoxのlsコマンドの優先度を200に変更できます。
(update-alternativesの使い方はHelpをご参照ください。)

[Module]# update-alternatives --install /bin/ls ls /bin/busybox.nosuid 200
update-alternatives: Linking /bin/ls to /bin/busybox.nosuid

200に変更されているのが確認できます。
[Module]# cat /usr/lib/opkg/alternatives/ls
/bin/ls
/bin/ls.coreutils 100
/bin/busybox.nosuid 200

Helpを見るとBusyBoxのlsになっているのがわかります。
[Module]# ls --help
BusyBox v1.31.1 () multi-call binary.

Usage: ls [-1AaCxdLHRFplinshrSXvctu] [-w WIDTH] [FILE]...

Linuxでは同じコマンド名でも動作が異なることがあります。
使用しているコマンドがどのレシピのコマンドなのか注意する必要があります。

RootFSのカスタマイズ

RootFSの修正はOSイメージを作成して出力されたRootFSを直接修正することでも可能です。OSイメージに含まれるRootFSを解凍して修正後、再度圧縮してください。下記の例は自作レシピ同様OS起動後、時刻のログを出力するシェルを自動実行するように修正します。

rootfs**修正用ディレクトリ作成**

[Ubuntu]$ cd /work/oe-core/image

[Ubuntu]$ sudo mkdir rootfs

[Ubuntu]$ cd rootfs

rootfs**解凍**

[Ubuntu]$ sudo tar -xvf ../Verdin-iMX8MP_Reference-Multimedia-Image-Tezi_<version>/Reference-Multimedia-Image-verdin-imx8mp.tar.xz

アプリケーションの作成

[Ubuntu]$ sudo gedit ./home/root/app.sh

app.sh**の内容**

- -

#!/bin/sh

date >> /home/root/log

- -

実行権限付与

[Ubuntu]$ sudo chmod +x ./home/root/app.sh

アプリケーション自動実行サービスの作成
[Ubuntu]$ sudo gedit ./etc/systemd/system/app.service

app.service**の内容**
- -
[Unit]
Description = Application

[Service]
ExecStart = /home/root/app.sh
Type = simple

[Install]
WantedBy = multi-user.target
- -

app.service**を有効にするためにシンボリックリンク作成**
[Ubuntu]$ cd ./etc/systemd/system/multi-user.target.wants/
[Ubuntu]$ sudo ln -s ../app.service app.service
[Ubuntu]$ cd ../../../../

元のファイルをリネームしバックアップ(任意)

[Ubuntu]$ sudo mv ../Verdin-iMX8MP_Reference-Multimedia-Image-Tezi_<version>/Reference-Multimedia-Image-verdin-imx8mp.tar.xz ../Verdin-iMX8MP_Reference-Multimedia-Image-Tezi_<version>/ReferenceMultimedia-Image-verdin-imx8mp_org.tar.xz

圧縮して戻します

[Ubuntu]$ sudo tar -Jcvf ../Verdin-iMX8MP_ReferenceMultimedia-Image-Tezi_<version>/ReferenceMultimedia-Image-verdin-imx8mp.tar.xz ./

Verdin-iMX8MP_Reference-Multimedia-Image-Tezi_<version>**配下のファイルを書き込めば起動するたびに**/home/root/log**に時刻が出力されます。**

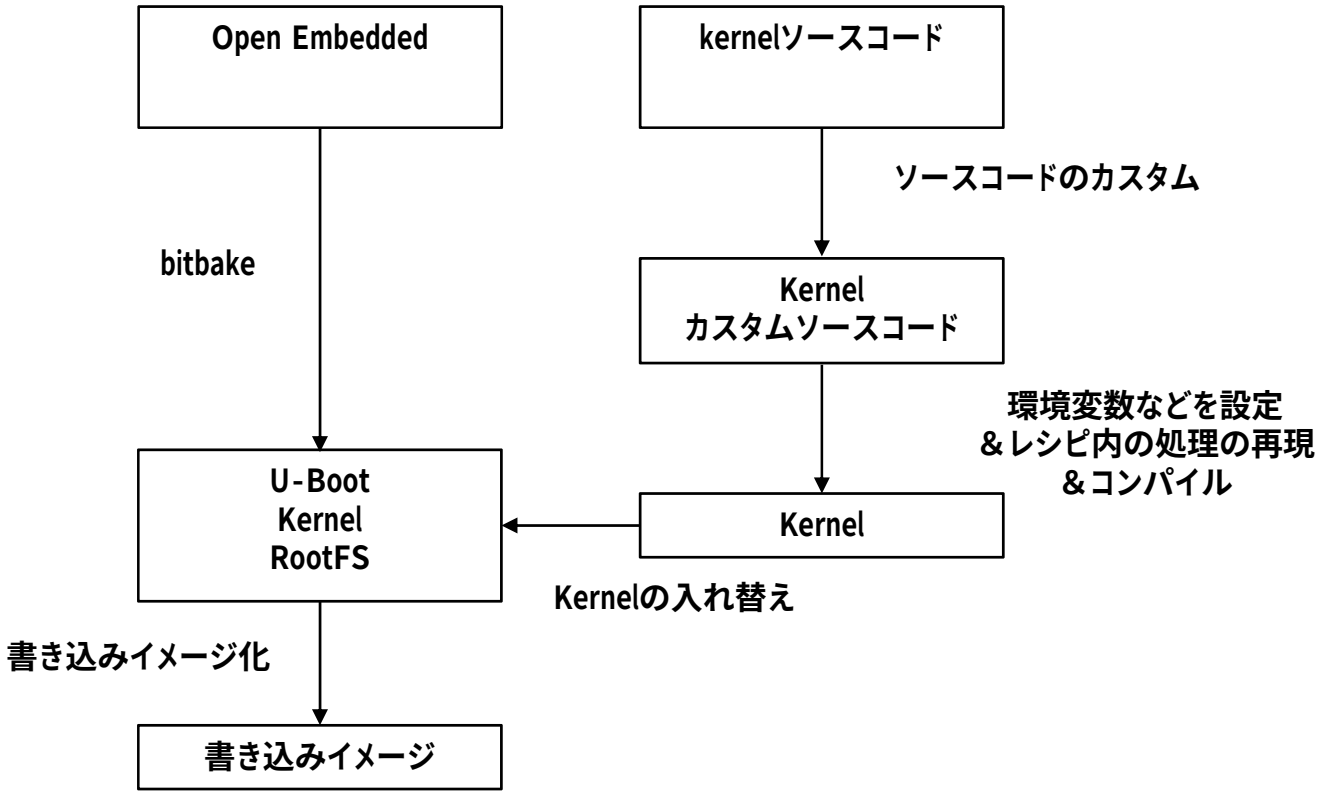
RootFSのカスタマイズの問題

RootFSを直接修正する方法は開発環境(パソコン)と実行環境(モジュール)が異なることによる問題が発生することがあります。
例えばファイルやディレクトリを作成すると権限はUbuntu上のユーザーの権限が付与されています。
これをそのまま実行環境にもっていくと存在しないユーザーの権限が付与されたファイルやディレクトリとなります。

またRootFSを直接編集する方法はbitbakeのたびに毎回手動で行う必要があります。
Open EmbeddedでRootfsを出力する場合はこのような問題は起きません。
できるだけレシピに組み込んで自動化する方が好ましいです。

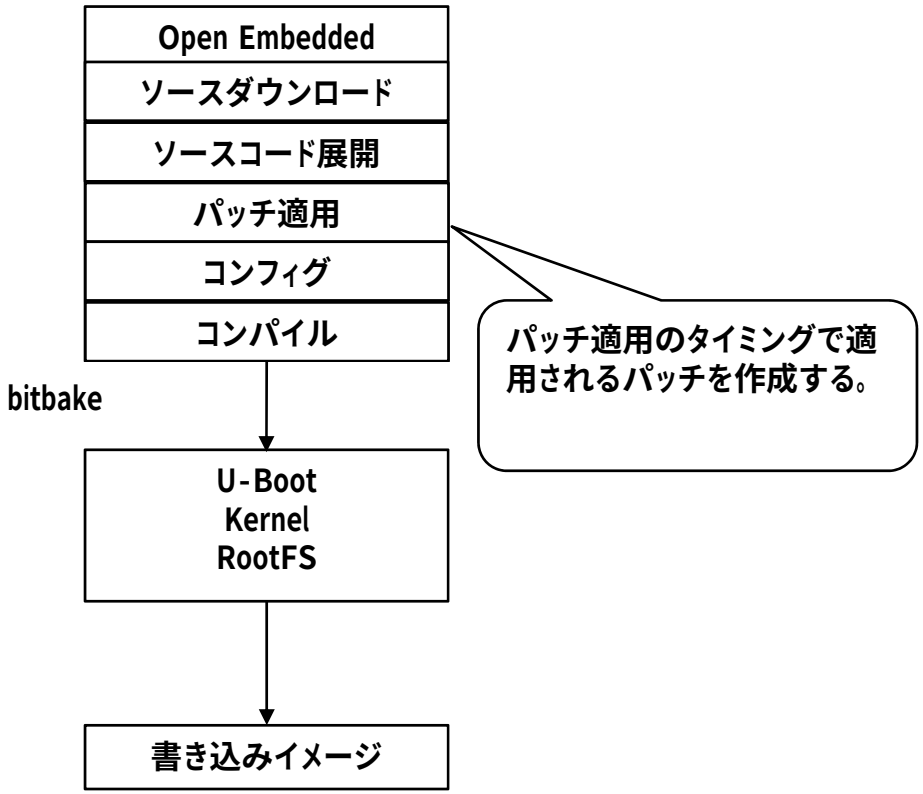
カーネルのカスタマイズ

カーネルのカスタマイズはカスタムする内容によりますが方法がいくつか考えられます。インターネット上には情報としてよくカーネルのソースコードをダウンロードして個別でコンパイルする方法の記載があります。個別でコンパイルするにはいくつかの注意を必要とします。特に注意しないといけないのがレシピ上では単純なコンパイル作業以外にもパッチを当てたりファイルを追加したり、バージョンを設定ファイルに書き込むなどさまざまな追加処理が存在します。個別でカーネルをコンパイルする場合、元のレシピの処理を読み解いて再現する必要があります。本手法はメンテナンス製が悪くレシピを読み解けなければできません。レシピの処理が多い場合は困難ですのであまり推奨はしません。



Open Embeddedに適したカーネルのカスタマイズ

bitbakeの一連の流れはソースコードのダウンロード、展開、パッチ適用、コンフィグ、コンパイルとなっています。
本マニュアルでのカスタマイズはパッチ適用のタイミングで適用されるパッチを作成することによりカーネルへの変更を加えます。



例としてカーネルの起動中のロゴと液晶の解像度を10.1インチLCD(LT170410)に変更するカスタムを行います。

下記コマンドでカーネルコンフィグするための機能をインストールしておきます。
[Ubuntu]$ sudo apt-get -y install libncurses5-dev

最初にLinuxのロゴで使用するppm形式のロゴファイルを作成します。Linuxのロゴはppm形式しか使えません。

netpbmをインストール
[Ubuntu]$ sudo apt-get -y install netpbm

ロゴにするBMPファイルを用意しLinuxのロゴ用のフォーマットに変換します。
本マニュアルではtest.bmpという1280x800(LCDの解像度)の画像ファイルを使用します。

ワーキングディレクトリ作成、移動
[Ubuntu]$ mkdir -p /work/patch/kernel
[Ubuntu]$ cd /work/patch/kernel

bmpからppmへ変換
[Ubuntu]$ bmptopnm test.bmp > test.ppm

224色へ減色
[Ubuntu]$ ppmquant 224 test.ppm > test_224.ppm

ASCIIへ変換
[Ubuntu]$ pnmnoraw test_224.ppm > logo_custom_clut224.ppm

devshell

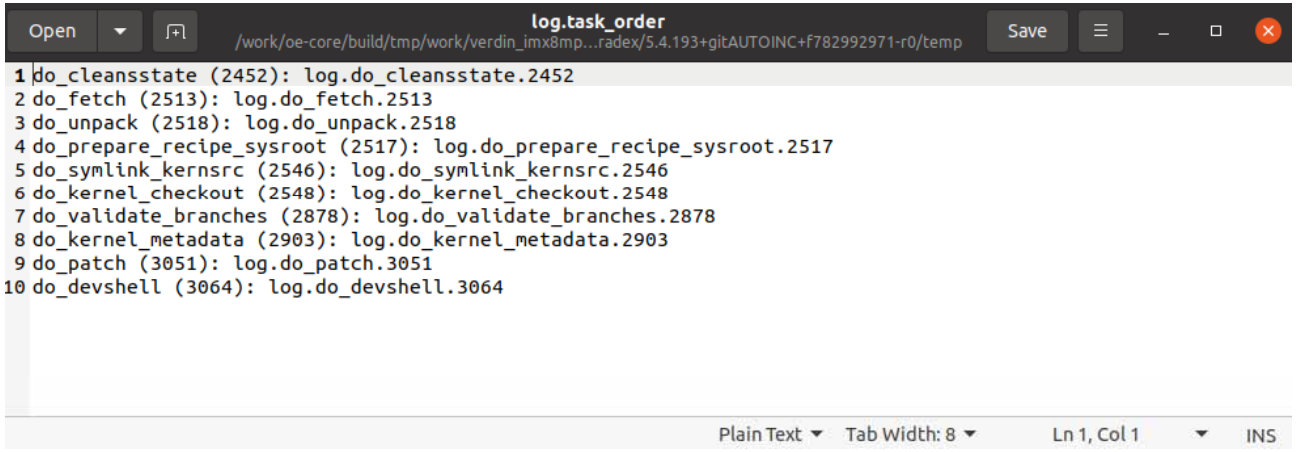
Open Embeddedではカスタムを容易にするためのdevshellという機能が用意されています。
OSをカスタムするためのパッチはdevshellを使用して作成します。

Open Embeddedの準備
[Ubuntu]$ cd /work/oe-core/
[Ubuntu]$. export

カーネルの初期化
[Ubuntu]$ bitbake -c cleansstate virtual/kernel

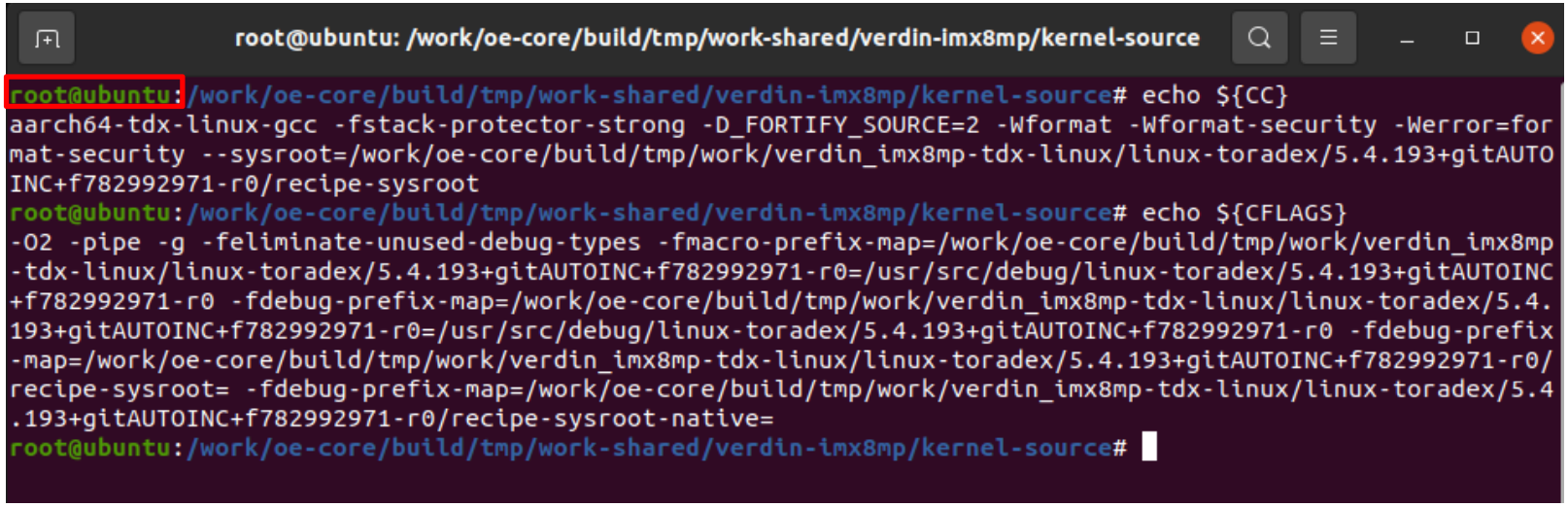
devshell実行
[Ubuntu]$ bitbake -c devshell virtual/kernel

カーネルのソースコードを展開したワーキングディレクトリで新しいターミナルが開きます。
本コマンドを実行した後にlog.task_order を確認するとfetch unpack patchなどが行われてからdevshellが行われていることがわかります。



devshellはbitbake時に設定される環境変数を再現してくれます。
例えば下記のようにCCやCFLAGSにはARM向けクロスコンパイラおよび必要なオプションが定義されています。

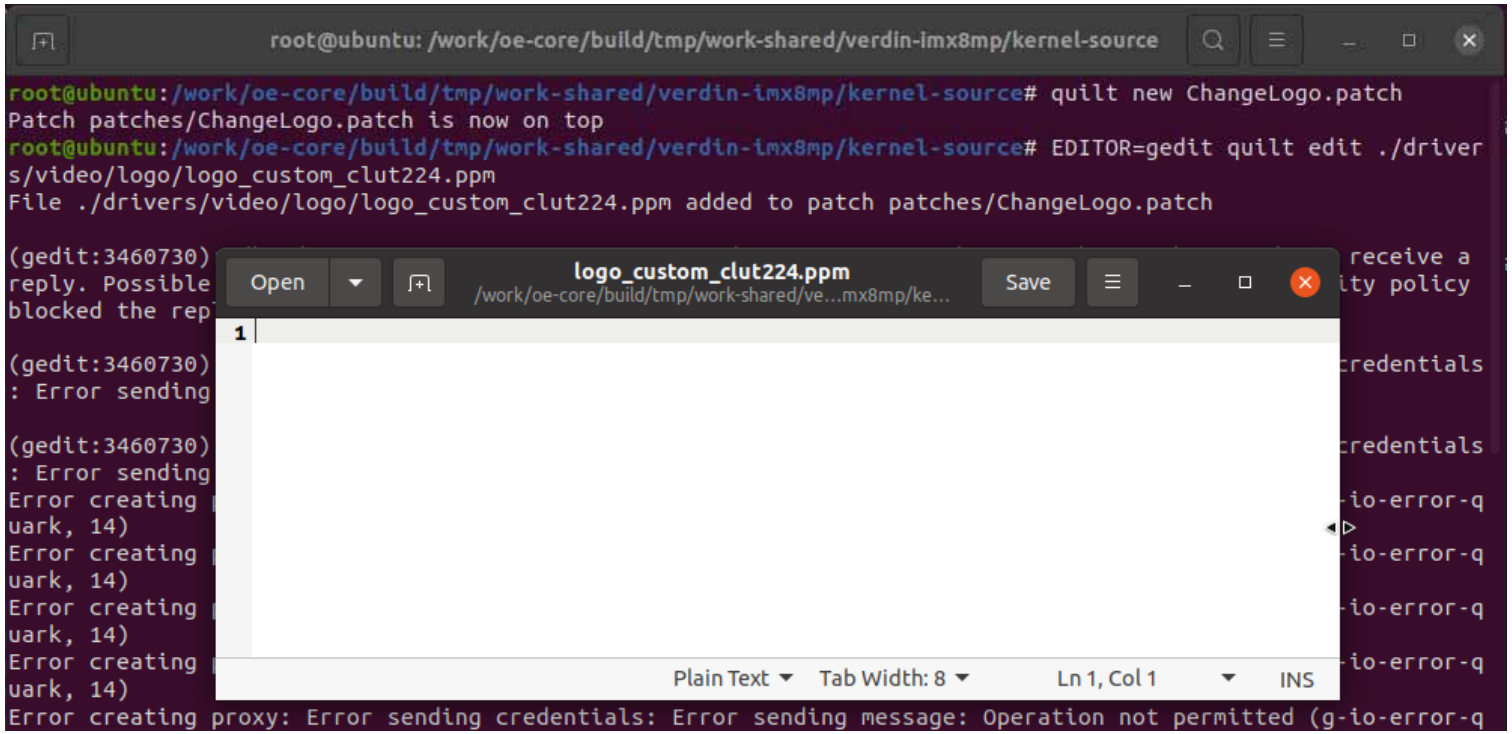
注：devshellを実行するとユーザー名がrootになります。これはfakerootで疑似的にrootになったような挙動をしているだけで実際にrootでログインしているわけではありません。



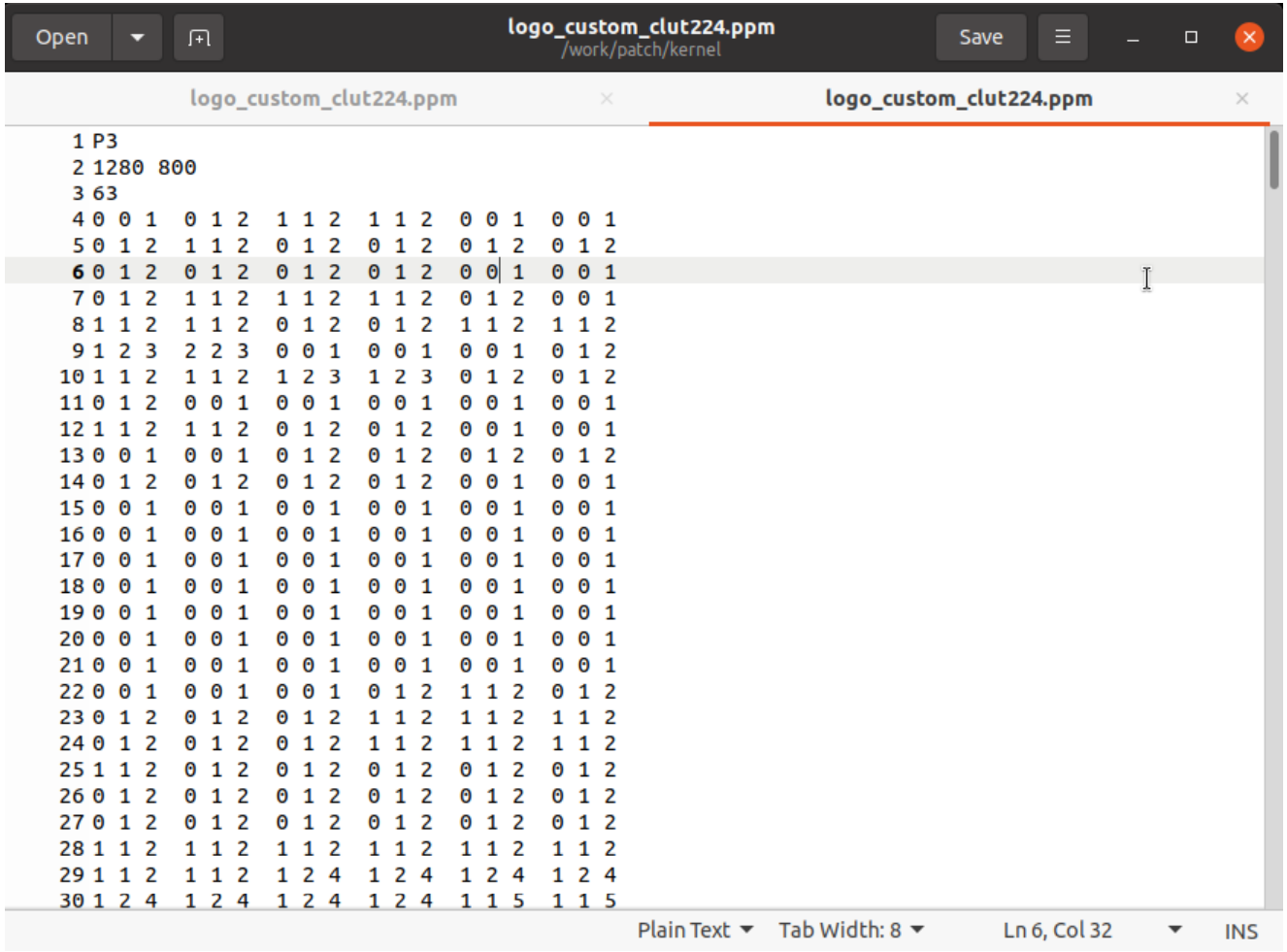
quiltコマンドを使ってカーネルのパッチを作ります。

パッチファイルの作成
[Ubuntu]$ quilt new ChangeLogo.patch

エディターをgeditに指定してlogo_custom_clut224.ppmを編集します。
[Ubuntu]$ EDITOR=gedit quilt edit ./drivers/video/logo/logo_custom_clut224.ppm



geditのメニューのOpenから最初に作成した/work/patch/kernel/logo_custom_clut224.ppmを開くと隣のタブに表示されます。
全選択してコピーして左のタブ側のlogo_custom_clut224.ppmに貼り付けてgeditを閉じます。



パッチの適用
[Ubuntu]$ quilt refresh

これでChangeLogo.patch**ができます。**

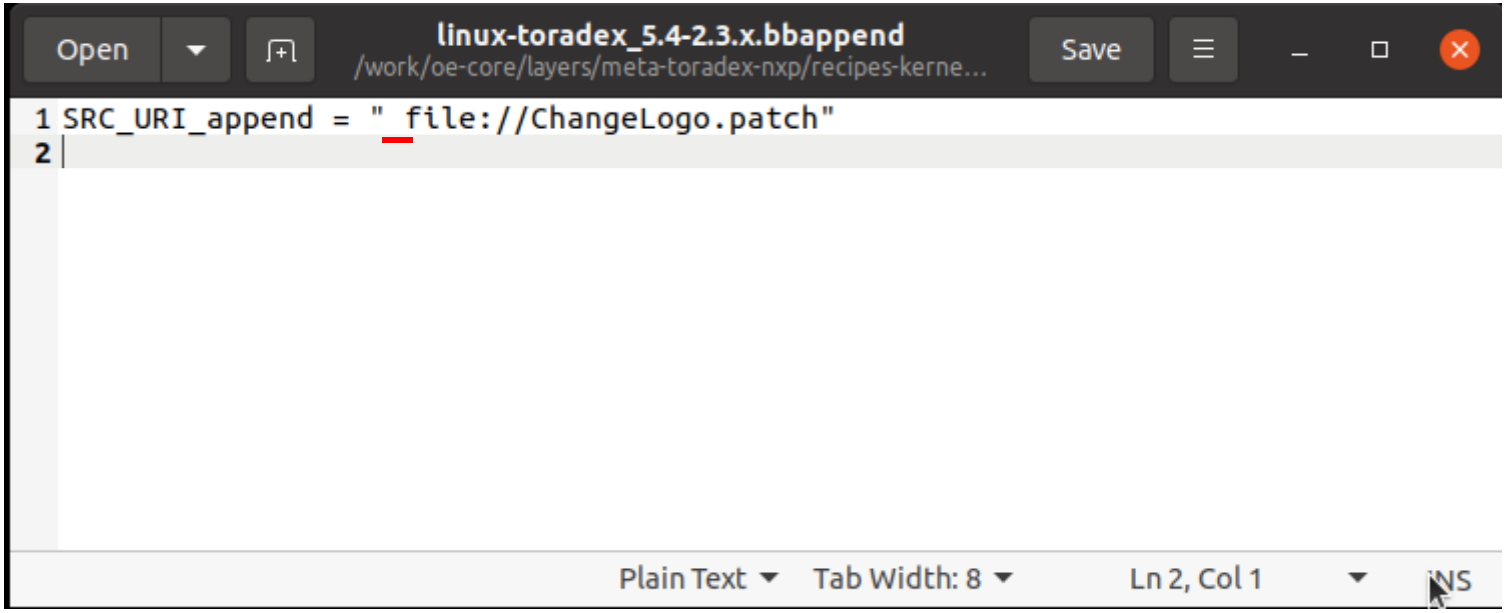
パッチファイルをカーネルのレシピに反映するために移動します。
[Ubuntu]$ mv ./patches/ChangeLogo.patch /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/linux-toradex-5.4-2.3.x/

dvshell**を終了します。**devshell**のターミナルが終了し**bitbake**を行ったターミナルに変わります。**
[Ubuntu]$ exit

作成したパッチファイルをカーネルのレシピに追加します。
拡張子bbappendファイルは同名bbファイルに追記するファイルです。もともとは存在しないため新規作成します。
[Ubuntu]$ gedit /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/linux-toradex_5.4-2.3.x.bbappend

追記内容
SRC_URI_append = " file://ChangeLogo.patch"

_appendとすると全モジュールに追記されますが_append_verdin-imx8mpとすることでVerdin-iMX8MPのみに適用するパッチになります。ロゴの修正は全モジュールに共通して変更しても問題ない内容なので_appendのみとします。
SRC_URI_appendはSRC_URIと文字列を結合します。そのためfile://ChangeLogo.patchの手前にスペースが必要です。
SRC_URI += という記述も可能ですがこちらは適用先を限定するような記述ができないため本マニュアルでは_appendを使用します。



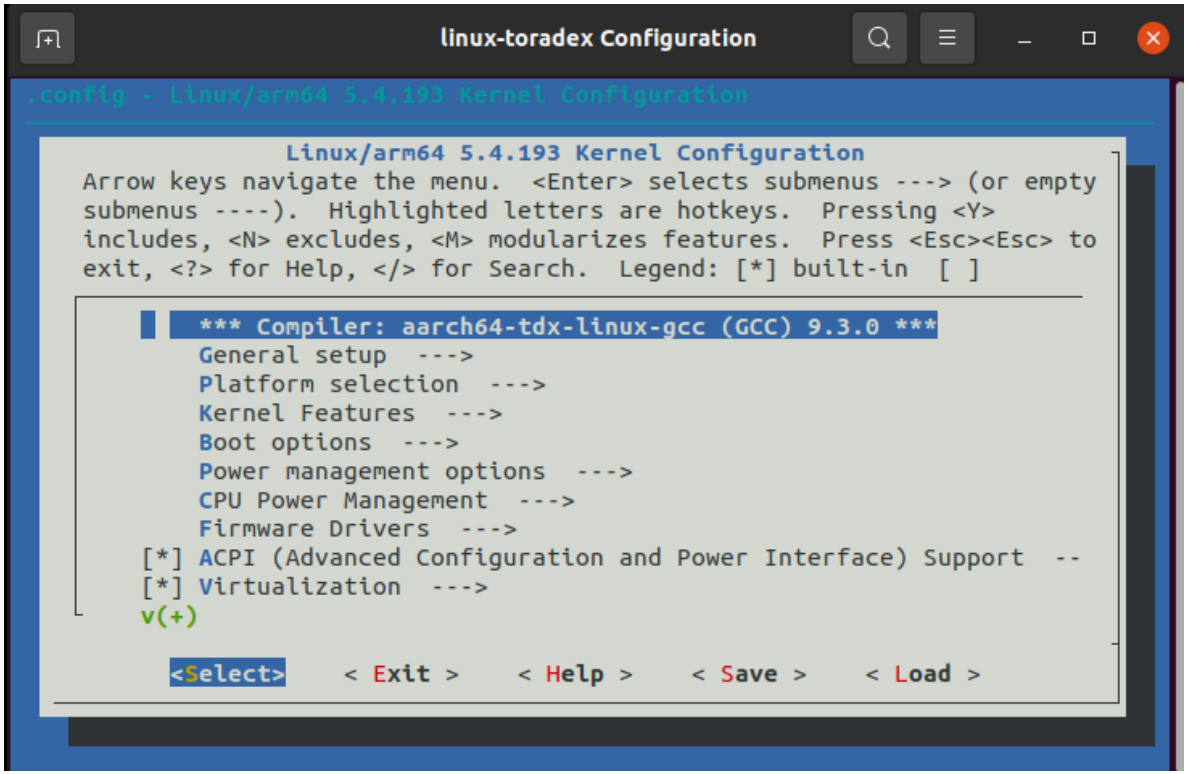
カーネルコンフィグ

ロゴを適用するにはカーネルのロゴの設定をカスタムロゴを使用するように変更する必要があります。

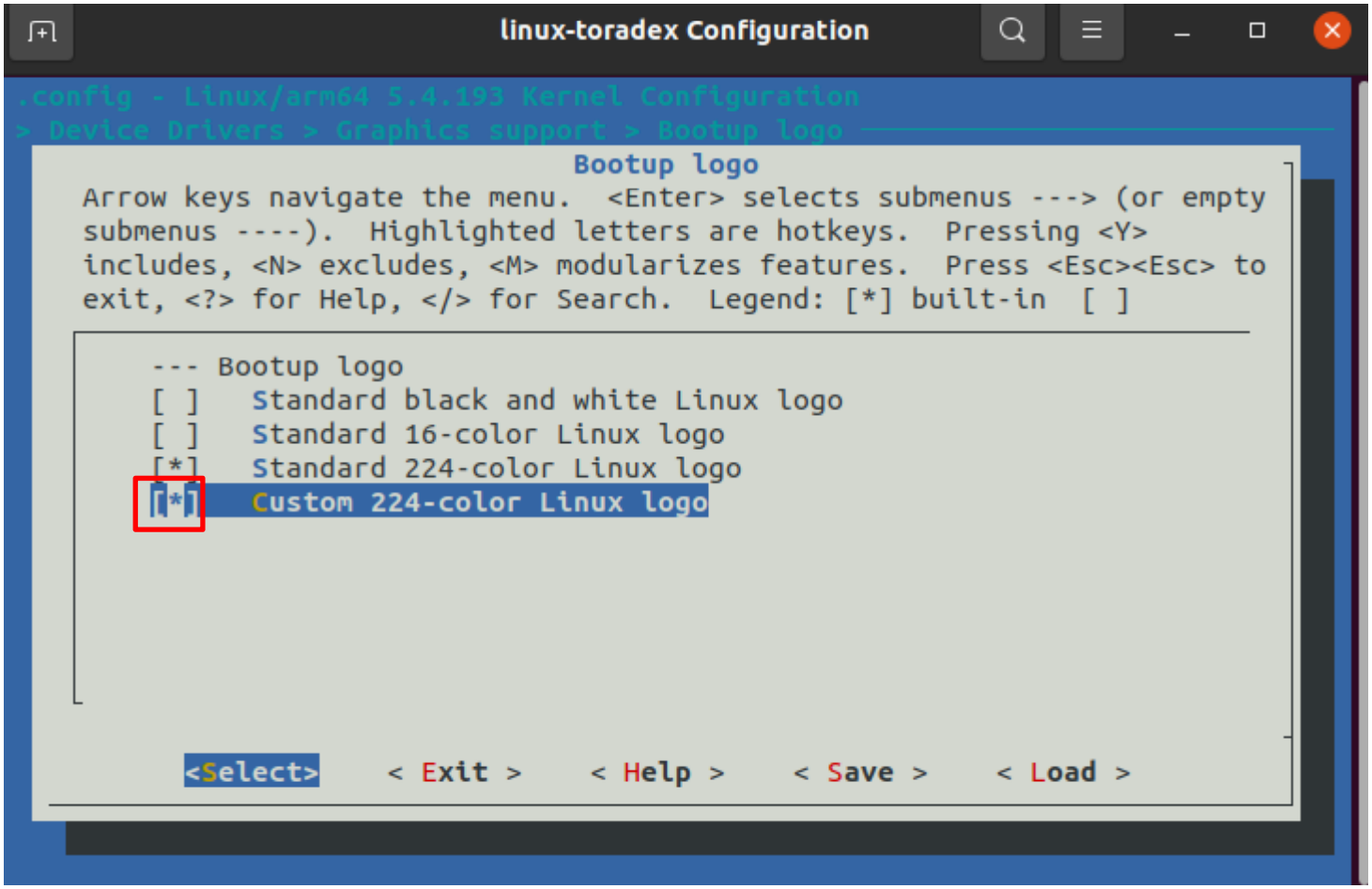
最初に初期化します。(先ほどのカーネルのカスタムを元に戻します。)
[Ubuntu]$ bitbake -c cleansstate virtual/kernel

下記コマンドを実行するとカーネルコンフィグが行えます。
[Ubuntu]$ bitbake -c menuconfig virtual/kernel

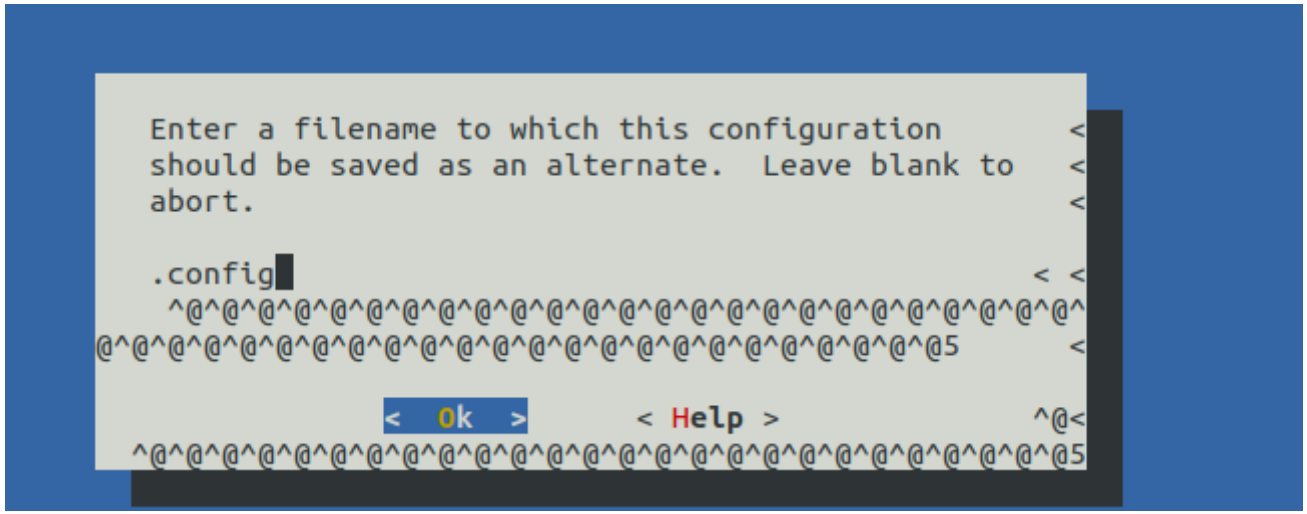
下記のような画面が起動します。



Device Drivers -> Graphics support -> Bootup logoを選択します。
スペースを入力してCustom 224-color Linux logoを有効にします。(カーネルコンフィグの操作に関しては省略します。)



画面下部の矢印キーでSaveを選択してファイル名はそのままでOKクリックします。
その後Exitを選択して保存します。画面下部の矢印キーでExitを選択してカーネルコンフィグを抜けます。



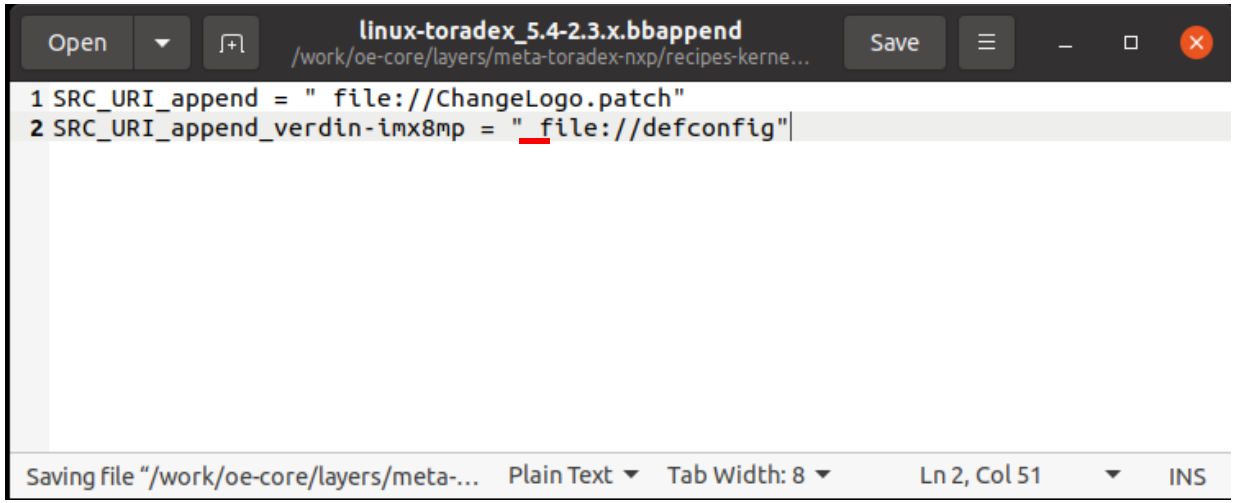
先ほど作成した.configをdefconfigとしてコピーします。
.configの所在については下記のコマンドで検索してください。
[Ubuntu]$ find /work/oe-core/build/tmp/work -name .config

defconfigを格納するディレクトリを作成します。
local.confにMACHINEで指定された名前で作成します。(verdin-imx8mp)
[Ubuntu]$ mkdir /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/linux-toradex-5.4-2.3.x/verdin-imx8mp

[Ubuntu]$ cp /work/oe-core/build/tmp/work/verdin_imx8mp-tdx-linux/linux-toradex/5.4.193+gitAUTOINC+f782992971-r0/build/.config /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/linux-toradex-5.4-2.3.x/verdin-imx8mp/defconfig

カーネルのレシピにdefconfigを追加
[Ubuntu]$ gedit /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/linux-toradex_5.4-2.3.x.bbappend

SRC_URI_append_verdin-imx8mp = " file://defconfig"



local.confに下記を追加します。
KBUILD_DEFCONFIG_verdin-imx8mp = ""

カーネルの初期化(一度編集した内容を元に戻します。)
[Ubuntu]$ bitbake -c cleansstate virtual/kernel

イメージの作成
[Ubuntu]$ bitbake tdx-reference-multimedia-image

出力されたファイルからOSイメージを作成してモジュールに書き込みます。

[Ubuntu]$ cd /work/oe-core/image/
[Ubuntu]$ sudo tar -xf ../build/deploy/images/apalis-imx8/Verdin-iMX8MP_Reference-Multimedia-Image-Tezi_<version>.tar

Linux起動時のブートロゴがカスタムしたものに変更されていることが確認できます。
ブートロゴはLinuxが起動してQTのサンプルアプリが動作するまでの少しの間しか表示されません。

OSの書き込み方法に関してはTEZIのマニュアルをご参照ください。

使用しない機能をGPIOに変更する例

カーネルのカスタムではデバイスツリーの修正を行うケースが多々あるためデバイスツリーの修正例についても記載します。
デバイスツリーはピンのコンフィグやドライバーの指定や設定などを設定するものです。
デバイスツリーはLinuxの基本的な仕組みのため詳細な説明については省きます。

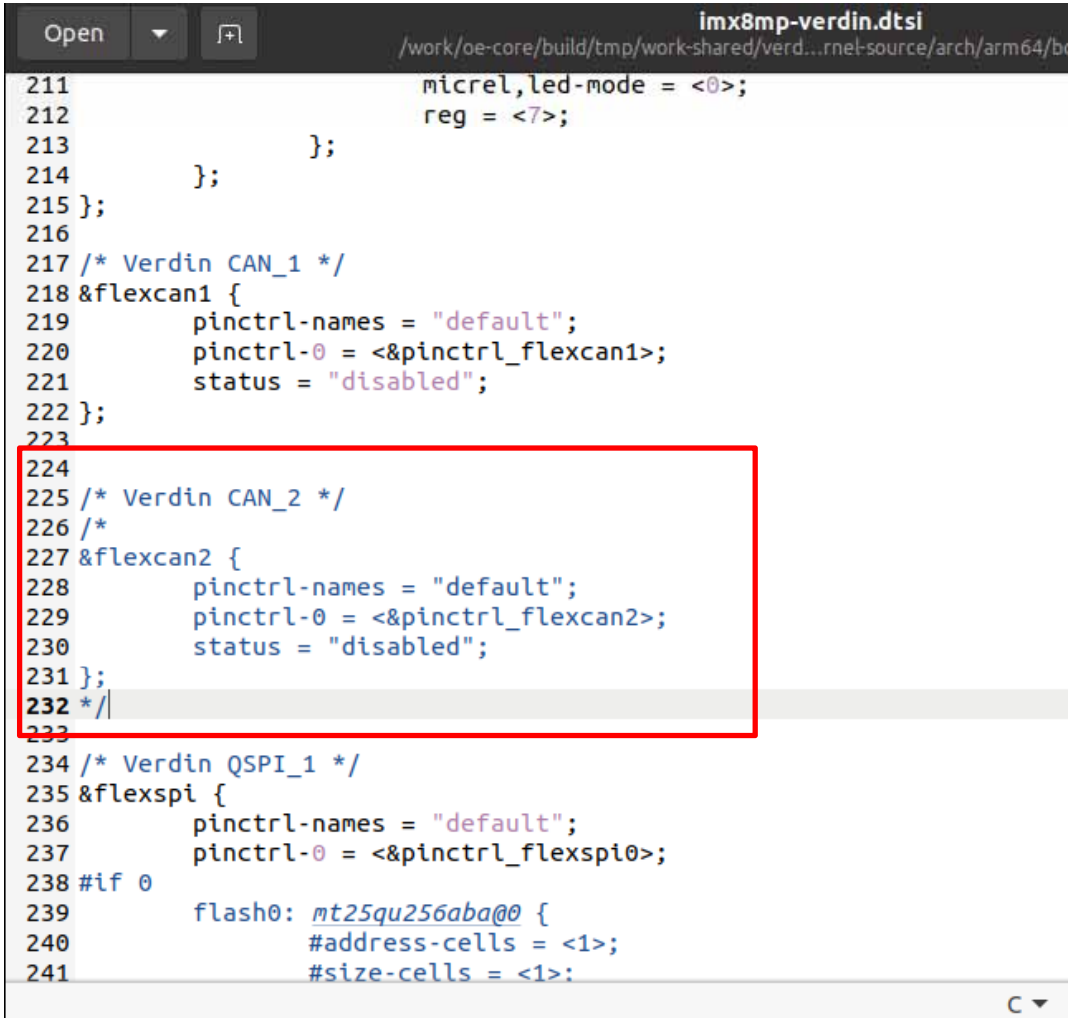
GPIOの端子を増やしたい場合には使用しない機能の端子をGPIOに変えて使用することがよくあります。
例えば24、26番ピンはCAN2に割り当てられています。CAN2を使用しない場合GPIOに変えて使用することができます。
CAN2に限らず他の機能でも同様です。ToradexのCPUモジュールはシリーズごとに互換性がありますが互換性がある端子をGPIOに変更する場合に限り、互換性を維持することができます。互換性の維持に関して詳しくはデザインガイドをご参照ください。

デバイスツリーを修正するためにカーネルのdevshellを行います。
[Ubuntu]$ bitbake -c devshell virtual/kernel

パッチファイルの作成
[Ubuntu]$ quilt new CAN2_GPIO.patch

デバイスツリーを修正します。
[Ubuntu]$ EDITOR=gedit quilt edit ./arch/arm64/boot/dts/freescale/imx8mp-verdin.dtsi

226行目辺りにあるCAN2の設定&flexcan2をコメントアウトします。



1006**行目辺りにある**CAN2**の設定**pinctrl_flexcan2**の端子設定を変更します。**

```
MX8MP_IOMUXC_SAI2_MCLK__CAN2_RX                    0x154        /* SODIMM 26 */
MX8MP_IOMUXC_SAI2_TXD0__CAN2_TX                    0x154        /* SODIMM 24 */
->
MX8MP_IOMUXC_SAI2_MCLK__GPIO4_IO27                 0x184        /* SODIMM 26 */
MX8MP_IOMUXC_SAI2_TXD0__GPIO4_IO26                 0x184        /* SODIMM 24 */
```

端子設定の説明をします。
./arch/arm64/boot/dts/freescale/imx8mp-pinfunc.hに各端子のピンコンフィグ設定が定義されています。
SODIMM 26の端子は572～578行目のMX8MP_IOMUXC_SAI2_MCLK__で始まる定義になります。
この定義は各端子のモード設定の数だけ存在します。上から順に0~6のモード設定になります。
設定値が5つ羅列されていますがMX8MP_IOMUXC_SAI2_MCLK__GPIO4_IO27の場合それぞれの意味は下記になります。
1.モード設定レジスタのオフセットアドレス　0x1B4　(MUX Control Register)
2.端子設定レジスタのオフセットアドレス　0x414　(PAD Control Register)
3.入力DAISY設定レジスタのオフセットアドレス　0x000
4.モード設定レジスタの設定値　0x5
5.入力DAISY設定レジスタの設定値　0x0

Open　imx8mp-pinfunc.h
/work/oe-core/build/tmp/work-shared/verdi...ernel-source/arch/arm64/boot/dts/freescale

```
564 #define MX8MP_IOMUXC_SAI2_TXC__GPIO4_IO25                    0x1AC 0x40C 0x000 0x5 0x0
565 #define MX8MP_IOMUXC_SAI2_TXC__AUDIOMIX_PDM_BIT_STREAM01     0x1AC 0x40C 0x4C4 0x6 0x6
566 #define MX8MP_IOMUXC_SAI2_TXD0__AUDIOMIX_SAI2_TX_DATA00      0x1B0 0x410 0x000 0x0 0x0
567 #define MX8MP_IOMUXC_SAI2_TXD0__AUDIOMIX_SAI5_TX_DATA03      0x1B0 0x410 0x000 0x1 0x0
568 #define MX8MP_IOMUXC_SAI2_TXD0__ENET_QOS_1588_EVENT2_IN      0x1B0 0x410 0x000 0x2 0x0
569 #define MX8MP_IOMUXC_SAI2_TXD0__CAN2_TX                      0x1B0 0x410 0x000 0x3 0x0
570 #define MX8MP_IOMUXC_SAI2_TXD0__ENET_QOS_1588_EVENT2_AUX_IN  0x1B0 0x410 0x000 0x4 0x0
571 #define MX8MP_IOMUXC_SAI2_TXD0__GPIO4_IO26                   0x1B0 0x410 0x000 0x5 0x0
572 #define MX8MP_IOMUXC_SAI2_MCLK__AUDIOMIX_SAI2_MCLK           0x1B4 0x414 0x000 0x0 0x0
573 #define MX8MP_IOMUXC_SAI2_MCLK__AUDIOMIX_SAI5_MCLK           0x1B4 0x414 0x4F0 0x1 0x2
574 #define MX8MP_IOMUXC_SAI2_MCLK__ENET_QOS_1588_EVENT3_IN      0x1B4 0x414 0x000 0x2 0x0
575 #define MX8MP_IOMUXC_SAI2_MCLK__CAN2_RX                      0x1B4 0x414 0x550 0x3 0x1
576 #define MX8MP_IOMUXC_SAI2_MCLK__ENET_QOS_1588_EVENT3_AUX_IN  0x1B4 0x414 0x000 0x4 0x0
577 #define MX8MP_IOMUXC_SAI2_MCLK__GPIO4_IO27                   0x1B4 0x414 0x000 0x5 0x0
578 #define MX8MP_IOMUXC_SAI2_MCLK__AUDIOMIX_SAI3_MCLK           0x1B4 0x414 0x4E0 0x6 0x1
579 #define MX8MP_IOMUXC_SAI3_RXFS__AUDIOMIX_SAI3_RX_SYNC        0x1B8 0x418 0x000 0x0 0x0
580 #define MX8MP_IOMUXC_SAI3_RXFS__AUDIOMIX_SAI2_RX_DATA01      0x1B8 0x418 0x4DC 0x1 0x1
581 #define MX8MP_IOMUXC_SAI3_RXFS__AUDIOMIX_SAI5_RX_SYNC        0x1B8 0x418 0x508 0x2 0x2
582 #define MX8MP_IOMUXC_SAI3_RXFS__AUDIOMIX_SAI3_RX_DATA01      0x1B8 0x418 0x000 0x3 0x0
583 #define MX8MP_IOMUXC_SAI3_RXFS__AUDIOMIX_SPDIF1_IN           0x1B8 0x418 0x544 0x4 0x2
584 #define MX8MP_IOMUXC_SAI3_RXFS__GPIO4_IO28                   0x1B8 0x418 0x000 0x5 0x0
585 #define MX8MP_IOMUXC_SAI3_RXFS__AUDIOMIX_PDM_BIT_STREAM00    0x1B8 0x418 0x4C0 0x6 0x5
586 #define MX8MP_IOMUXC_SAI3_RXC__AUDIOMIX_SAI3_RX_BCLK         0x1BC 0x41C 0x000 0x0 0x0
587 #define MX8MP_IOMUXC_SAI3_RXC__AUDIOMIX_SAI2_RX_DATA02       0x1BC 0x41C 0x000 0x1 0x0
588 #define MX8MP_IOMUXC_SAI3_RXC__AUDIOMIX_SAI5_RX_BCLK         0x1BC 0x41C 0x4F4 0x2 0x2
589 #define MX8MP_IOMUXC_SAI3_RXC__GPT1_CLK                      0x1BC 0x41C 0x59C 0x3 0x0
590 #define MX8MP_IOMUXC_SAI3_RXC__UART2_DCE_CTS                 0x1BC 0x41C 0x000 0x4 0x0
591 #define MX8MP_IOMUXC_SAI3_RXC__UART2_DTE_RTS                 0x1BC 0x41C 0x5EC 0x4 0x2
592 #define MX8MP_IOMUXC_SAI3_RXC__GPIO4_IO29                    0x1BC 0x41C 0x000 0x5 0x0
593 #define MX8MP_IOMUXC_SAI3_RXC__AUDIOMIX_PDM_CLK              0x1BC 0x41C 0x000 0x6 0x0
```

これらの定義は変更する必要なくモード0~6のどれを使うかを定義を使って選択するだけです。
各モードに対してどのような値を入力するか考える必要なく、定義を選択するだけで使用できるようになっています。

NXPのリファレンスマニュアルの下記にあたります。(リファレンスマニュアルはNXPにアカウント登録して入手してください)
8.2.4.105 SW_MUX_CTL_PAD_SAI2_MCLK SW MUX Control Register

MX8MP_IOMUXC_SAI2_MCLK__GPIO4_IO27 0x184 /* SODIMM 26 */

上記のMX8MP_IOMUXC_SAI2_MCLK__GPIO4_IO27でモード5のGPIO設定を選んでいます。
0x184の部分は端子設定レジスタの設定値になります。

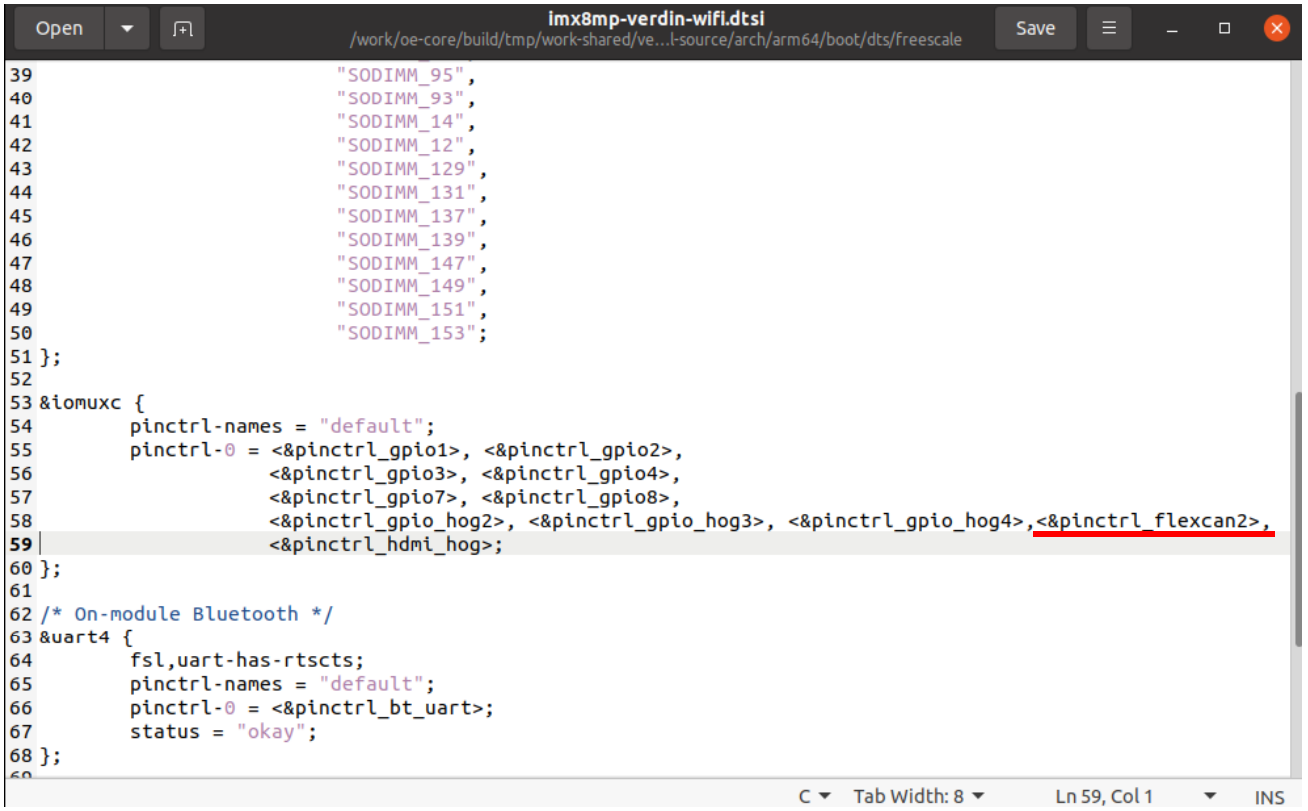
NXPのリファレンスマニュアルの下記にあたります。
8.2.4.257 SW_PAD_CTL_PAD_SAI2_MCLK SW PAD Control Register

0x184はプルダウン設定になります。

GPIOの設定として使う端子は&iomuxcに入れる必要があります。

[Ubuntu]$ EDITOR=gedit quilt edit ./arch/arm64/boot/dts/freescale/imx8mp-verdin-wifi.dtsi

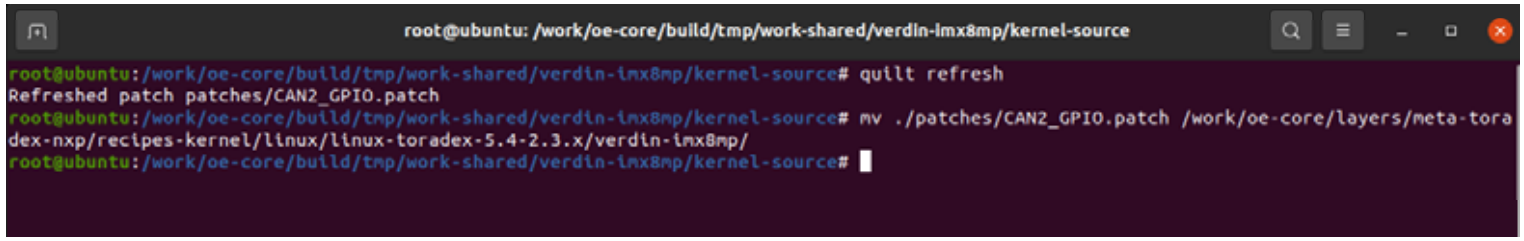
58行目に下記を追記します。
<&pinctrl_flexcan2>,



修正内容に対するパッチを作成します。
[Ubuntu]$ quilt refresh

CAN2_GPIO.patch**というファイル名でパッチファイルが出力されます。**

パッチファイルをカーネルのレシピに反映するために移動します。
[Ubuntu]$ mv ./patches/CAN2_GPIO.patch /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/linux-toradex-5.4-2.3.x/verdin-imx8mp/

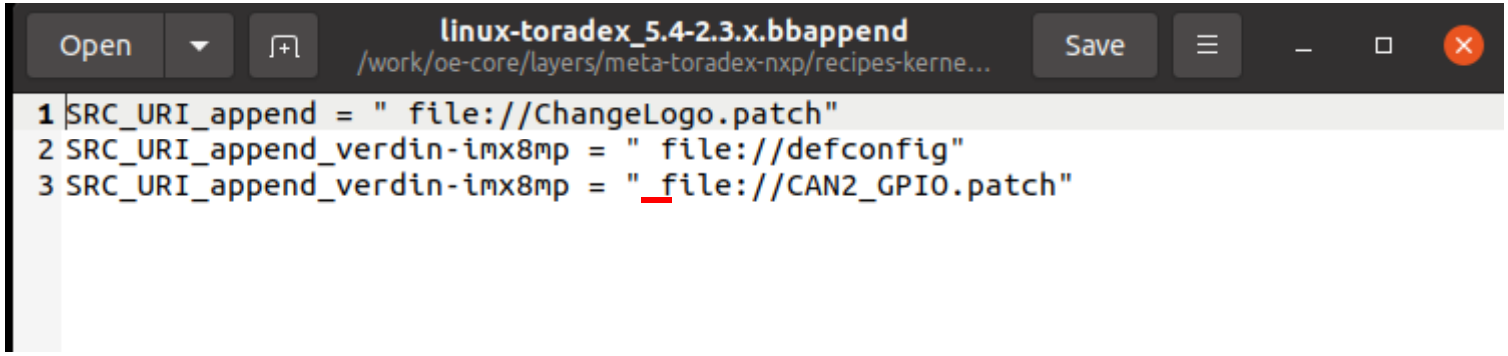


devshell**を終了します。**
[Ubuntu]$ exit

パッチファイルを追記します。
[Ubuntu]$ gedit /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/linux-toradex_5.4-2.3.x.bbappend

追記内容(verdin-imx8mp**のみに適用します。**)
SRC_URI_append_verdin-imx8mp = " file://CAN2_GPIO.patch"

イメージの作成
[Ubuntu]$ bitbake -c cleansstate virtual/kernel && bitbake tdx-reference-multimedia-image



出力されたファイルからOSイメージを作成してモジュールに書き込みます。

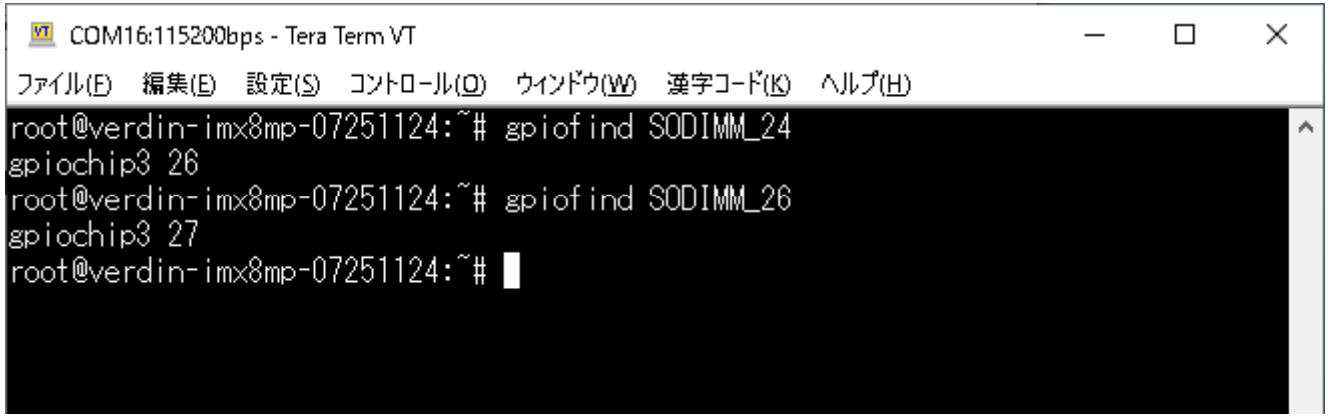
[Ubuntu]$ cd /work/oe-core/image/
[Ubuntu]$ sudo tar -xf ../build/deploy/images/apalis-imx8/Verdin-iMX8MP_Reference-Multimedia-Image-Tezi_<version>.tar

OSの書き込み方法に関してはTEZIのマニュアルをご参照ください。

書き込み後Linuxを起動してGPIOの動作確認をします。
CAN2 TXは24番ピン、CAN2 RXは26番ピンです。

gpiofindコマンドで端子を検索します。
[Module]# gpiofind SODIMM_24
[Module]# gpiofind SODIMM_26

gpiochip3 26はGPIOバンク3の26ラインという意味になります。
このバンクとラインはGPIOを指定するときに使用する値になります。

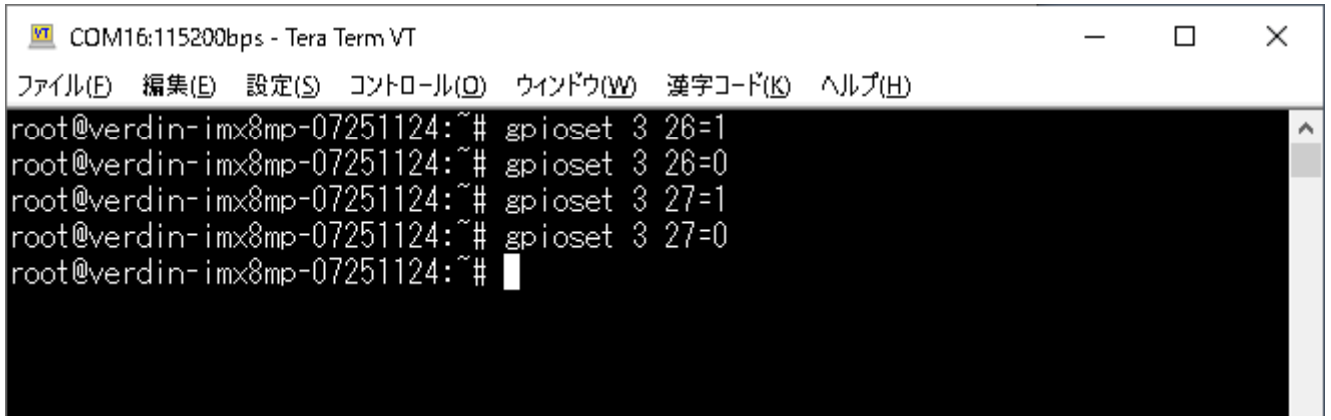


GPIOの出力を試します。
Verdin開発ボードのX2のSODIMM24(7番目)とX4のCAN2_TXを接続しているX3のジャンパーを外してください。
X2のSODIMM26(8番目)とX4のCAN2_RXを接続しているX3のジャンパーも同様に外してください。
X2のSODIMM24とX38のLED21(2番目)に接続します。

gpiosetコマンドでLEDのON/OFFを確認します。
LED ON
[Module]# gpioset 3 26=1
LED OFF
[Module]# gpioset 3 26=0

X2のSODIMM26も同様にX38のLED22(3番目)に接続して確認します。

LED ON
[Module]# gpioset 3 27=1
LED OFF
[Module]# gpioset 3 27=0



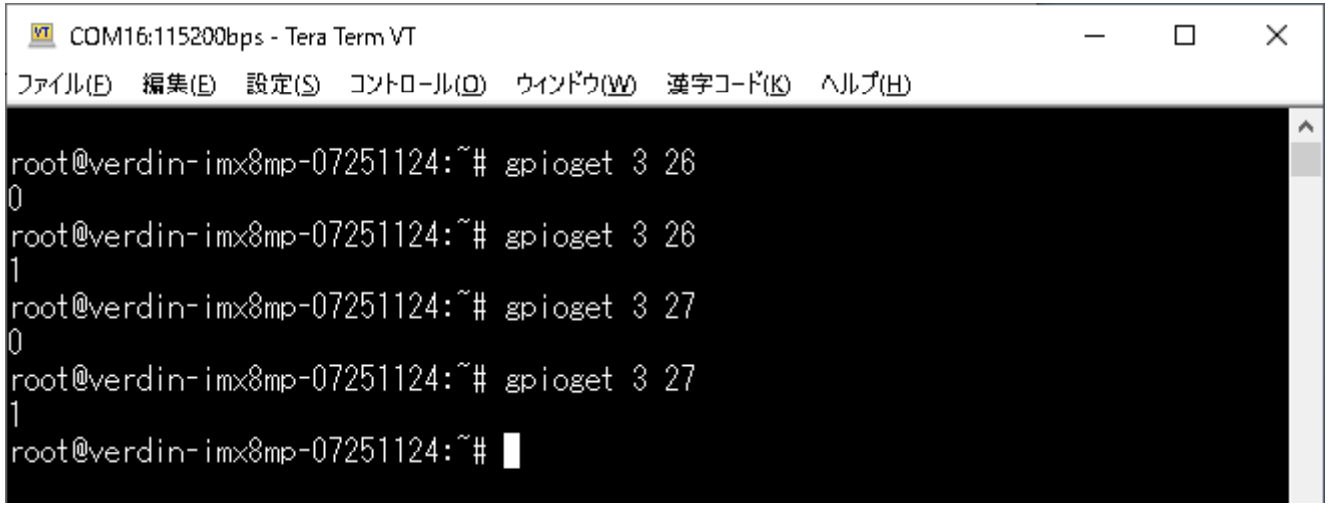
GPIOの入力を試します。
Verdin開発ボードのX2のSODIMM24とX4のCAN2_TXを接続しているX3のジャンパーを外してください。
X2のSODIMM26とX4のCAN2_RXを接続しているX3のジャンパーも同様に外してください。
X2のSODIMM24とX23のSW4に接続します。

gpiogetコマンドでSWの入力を確認します。

[Module]# gpioget 3 26

X2のSODIMM26も同様にX24のSW5に接続して確認します。

[Module]# gpioget 3 27



デバイスツリーオーバーレイのカスタム

Verdin開発ボードにVerdin DSI to LVDS Adapterと10.1インチLCD(LT170410)を接続して表示するためのカスタムを行います。

カスタムを行う前に予備知識としてToradexが用意しているデバイスツリーオーバーレイについて説明します。

デバイスツリーにはデバイスツリーオーバーレイという仕組みがあり、ブートローダーから指定されたデバイスツリーを読み込み後、
デバイスツリーオーバーレイファイルを読み込み部分的にデバイスツリーを上書きして設定などを変更することができます。
Linux起動中に/boot/overlays.txtを参照すると適用されているデバイスツリーオーバーレイが確認できます。
デフォルトではnativeのHDMIを使用するように下記のようになっています。

fdt_overlays=verdin-imx8mp_native-hdmi_overlay.dtbo verdin-imx8mp_lt8912_overlay.dtbo

上記はVerdin DSI to HDMI Adapterを動かすためのデバイスツリー
verdin-imx8mp_native-hdmi_overlay.dtbo
verdin-imx8mp_lt8912_overlay.dtbo
の2つ適用されています。
これをVerdin DSI to LVDS Adapterと10.1インチLCD(LT170410)を動かすデバイスツリーに変更します。
/boot/overlays.txtを下記に書き換えて再起動するとLCDが表示されるようになります。
fdt_overlays=touch-atmel-mxt_overlay.dtbo verdin-imx8mp_sn65dsi84-lt170410_overlay.dtbo verdin-imx8mp_sn65dsi84_overlay.dtbo

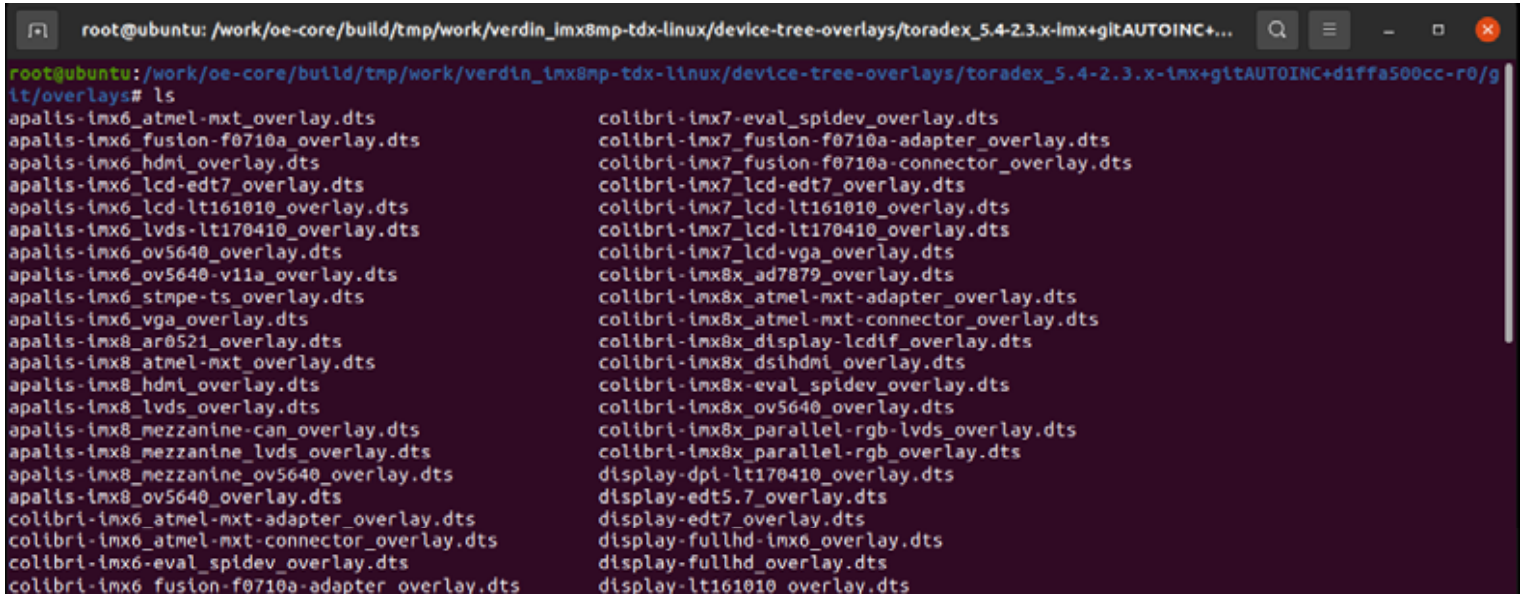
Verdin DSI to LVDS Adapter以外にもいくつかデバイスツリーオーバーレイファイルが用意されています。
何を指定すればよいかは下記に情報があります。
https://developer-archives.toradex.com/knowledge-base/setting-up-recommended-displays-with-torizon#Find_the_Correspondent_Device_Tree_Overlays

設定したデバイスツリーには下記3つが適用されています。
touch-atmel-mxt_overlay.dtbo
verdin-imx8mp_sn65dsi84-lt170410_overlay.dtbo
verdin-imx8mp_sn65dsi84_overlay.dtbo

拡張子dtbo**はテキストベースの拡張子**dts**ファイルをコンパイルしてバイナリ化されたファイルです。**
dts**ファイルを参照することによりどんな設定がされているかを確認できます。**
デバイスツリーオーバーレイはdevice-tree-overlays**のレシピで作成されています。**
devshell**でコンパイル前の**dts**ファイルを確認することができます。**

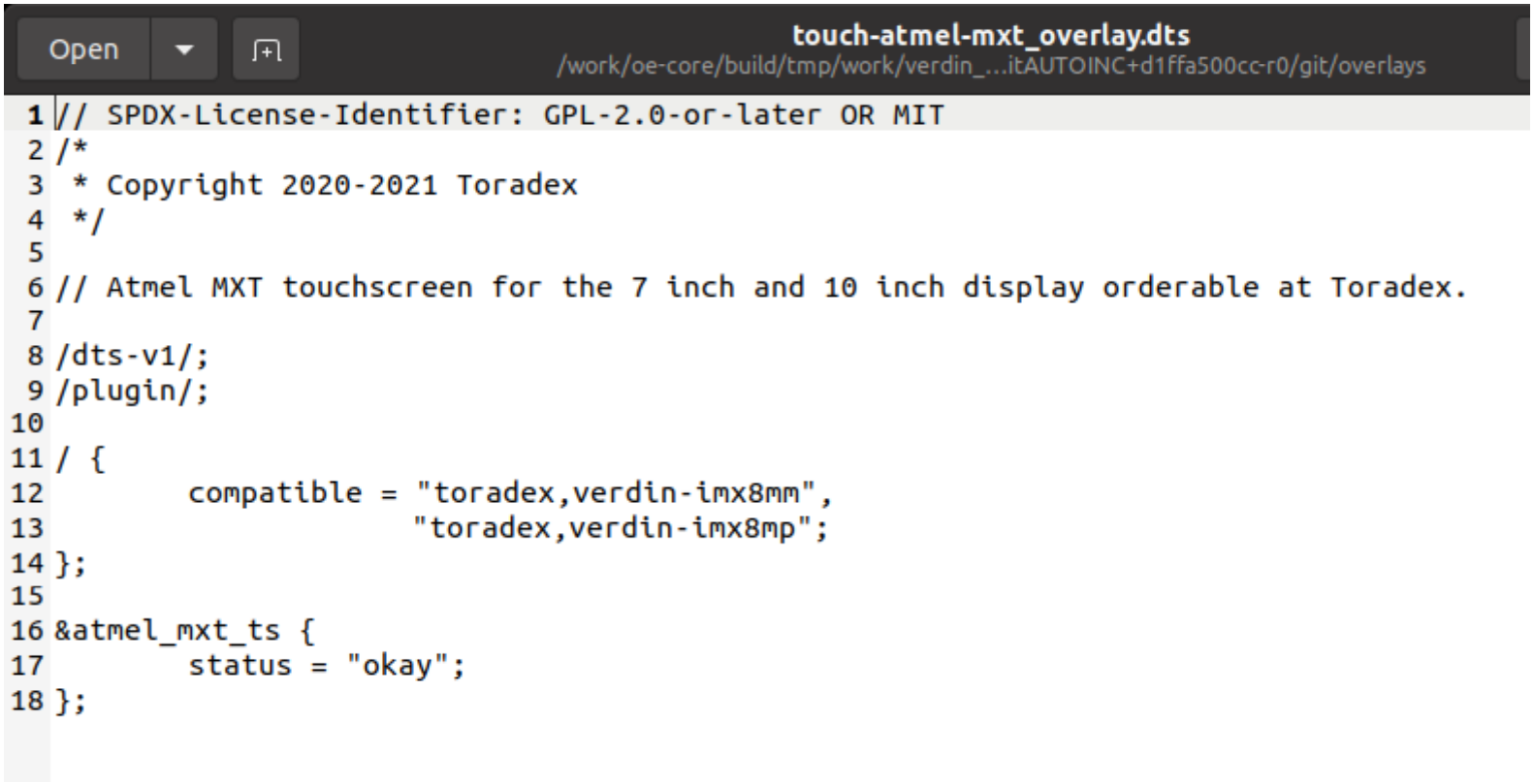
[Ubuntu]$ bitbake -c devshell device-tree-overlays

devshell**で開いた後、ファイルを確認すると拡張子**dts**のファイルがいくつかあります。**



各々のファイルを確認します。
[Ubuntu]$ gedit touch-atmel-mxt_overlay.dts

8〜9行目はDTS記述バージョンの指定とベースのデバイスツリーを参照するための記述です。
11〜14行目までは対応するハードウェアの指定です。
このファイルの設定部分は16〜18行目部分のみです。Atmel製のタッチパネルコントローラを有効に設定しています。



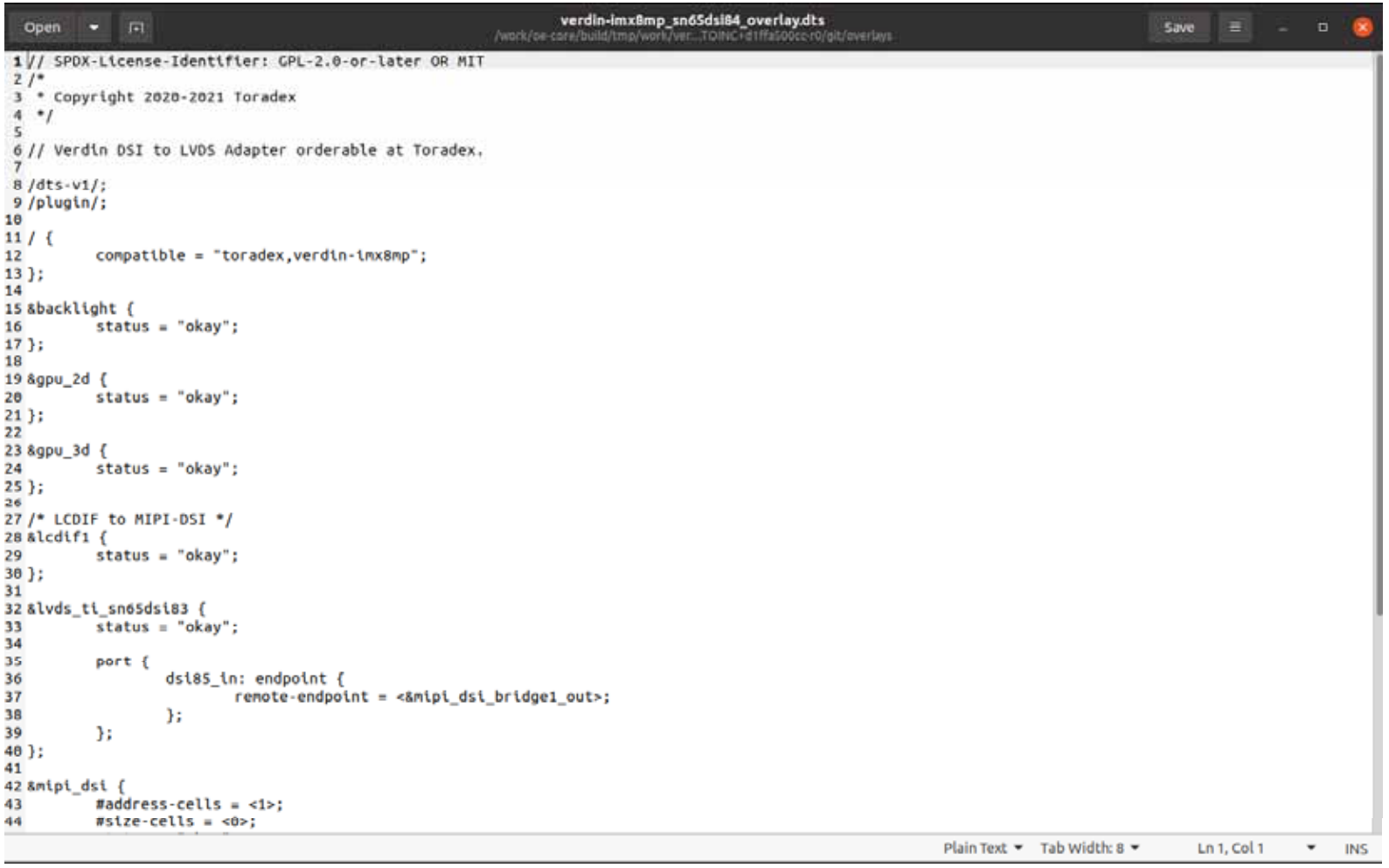
[Ubuntu]$ gedit verdin-imx8mp_sn65dsi84-lt170410_overlay.dts

このファイルの設定部分は16～44**目です。**
MIPI-DSI**を**LVDS**に変換する**IC(SN65DSI84)**の設定と**10.1**インチ**LCD(LT170410)**の設定を行っています。**
別のLCD**を使用する場合、**LCD**に応じて設定を変更してください。**

```
15
16 &lvds_ti_sn65dsi83 {
17         ti,dsi-lanes = <4>;
18         ti,height-mm = <136>;
19         ti,lvds-bpp = <24>;
20         ti,lvds-format = <2>;
21         ti,width-mm = <217>;
22
23         display-timings {
24                 native-mode = <&lvds_timing0>;
25
26                 lvds_timing0: lt170410_2whc {
27                         /*
28                          * PLL1 is at 2079000000, take PLL1/30
29                          * otherwise we don't get a picture on NXP i.MX 8M Plus
30                          */
31                         clock-frequency = <69300000>;
32                         hactive = <1280 1280 1280>;
33                         hfront-porch = <23 60 71>;
34                         hback-porch = <23 60 71>;
35                         hsync-len = <15 40 47>;
36                         vactive = <800 800 800>;
37                         vfront-porch = <5 7 10>;
38                         vback-porch = <5 7 10>;
39                         vsync-len = <6 9 12>;
40                         de-active = <1>;
41                         pixelclk-active = <0>;
42                 };
43         };
44 };
```

[Ubuntu]$ gedit verdin-imx8mp_sn65dsi84_overlay.dts

このファイルの設定部分は15～61**目です。**
必要な機能の有効化とMIPI-DSI**の設定を行っています。**




```
1 // SPDX-License-Identifier: GPL-2.0-or-later OR MIT
2 /*
3  * Copyright 2020-2021 Toradex
4  */
5
6 // Verdin DSI to LVDS Adapter orderable at Toradex.
7
8 /dts-v1/;
9 /plugin/;
10
11 / {
12         compatible = "toradex,verdin-imx8mp";
13 };
14
15 &backlight {
16         status = "okay";
17 };
18
19 &gpu_2d {
20         status = "okay";
21 };
22
23 &gpu_3d {
24         status = "okay";
25 };
26
27 /* LCDIF to MIPI-DSI */
28 &lcdif1 {
29         status = "okay";
30 };
31
32 &lvds_ti_sn65dsi83 {
33         status = "okay";
34
35         port {
36                 dsi85_in: endpoint {
37                         remote-endpoint = <&mipi_dsi_bridge1_out>;
38                 };
39         };
40 };
41
42 &mipi_dsi {
43         #address-cells = <1>;
44         #size-cells = <0>;
```

本来はLCDの設定を変えたりMIPI to LVDSを別のIC用に修正するような変更を行いますが
本マニュアルでは変更する意味はありませんがタッチパネルを無効にする修正を行います。
修正内容には意味はありませんがdevice-tree-overlaysの**修正例として参考にしてください。**

パッチファイルの作成
[Ubuntu]$ quilt new touch-disable.patch

下記ファイルにtouch-atmel-mxt_overlay.dts**と**verdin-imx8mp_sn65dsi84_overlay.dts**の設定を追記します。**
[Ubuntu]$ EDITOR=gedit quilt edit ./touch-atmel-mxt_overlay.dts

タッチパネルを無効にします。
status = "okay";
->
status = "disable";

修正内容に対するパッチを作成します。
[Ubuntu]$ quilt refresh

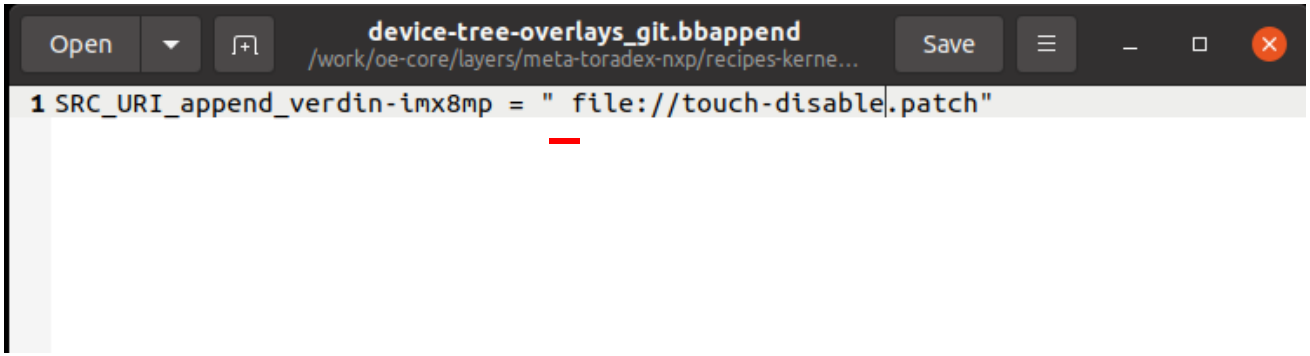
touch-disable.patch**というファイル名でパッチファイルが出力されます。**

パッチを置くディレクトリを作成します。
[Ubuntu]$ mkdir /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/device-tree-overlays

パッチファイルをレシピに反映するために移動します。
[Ubuntu]$ mv ./patches/touch-disable.patch /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/device-tree-overlay
s/

bbapend**ファイルを作成します。**
[Ubuntu]$ gedit /work/oe-core/layers/meta-toradex-nxp/recipes-kernel/linux/device-tree-overlays_git.bbappend

SRC_URI_append_verdin-imx8mp = " file://touch-disable.patch"



イメージの作成

[Ubuntu]$ bitbake -c cleansstate device-tree-overlays && bitbake tdx-reference-multimedia-image

出力されたファイルからOSイメージを作成してモジュールに書き込みます。

[Ubuntu]$ cd /work/oe-core/image/

[Ubuntu]$ sudo tar -xf ../build/deploy/images/apalis-imx8/Verdin-iMX8MP_Reference-Multimedia-Image-Tezi_<version>.tar

OS**の書き込み方法に関しては**TEZI**のマニュアルをご参照ください。**

Linux**起動後**/boot/overlay.txt**を書き換えます。**

fdt_overlays=verdin-imx8mp_native-hdmi_overlay.dtbo verdin-imx8mp_lt8912_overlay.dtbo

->

fdt_overlays=fdt_overlays=touch-atmel-mxt_overlay.dtbo verdin-imx8mp_sn65dsi84-lt170410_overlay.dtbo verdin-imx8mp_sn65dsi84_overlay.dtbo

再起動後タッチパネルが効かないことを確認できます。

U-Bootのカスタマイズ

カーネルのカスタマイズではロゴをカスタムしましたがロゴの下に起動ログが出力されます。またカーソルも表示されています。
本マニュアルではU-Bootのカスタム例としてカーネルに渡すカーネルパラメータを変更してこれらを取り除きます。
ブートローダーの仮想レシピ名はvirtual/bootloaderになります。

Open Embeddedの準備
[Ubuntu]$ cd /work/oe-core/
[Ubuntu]$. export

ブートローダーのレシピはbitbake -c checkuri virtual/bootloader > logで調査すると
/work/oe-core/build/../layers/meta-toradex-nxp/recipes-bsp/u-boot/u-boot-toradex_2020.04.bb
ということがわかります。

初期化
[Ubuntu]$ bitbake -c cleansstate virtual/bootloader

devshell実行
[Ubuntu]$ bitbake -c devshell virtual/bootloader

パッチファイルの作成
[Ubuntu]$ quilt new ChangeKernelParameter.patch

設定を変更します。
[Ubuntu]$ EDITOR=gedit quilt edit ./include/configs/verdin-imx8mp.h

setupの設定から
console=tty1(ディスプレイにコンソール出力を表示する設定)を削除して
vt.global_cursor_default=0(カーソルを消す)を追加します。

変更箇所は内容は下記です。
112行目あたり
"setup=setenv setupargs console=${console},${baudrate} console=tty1 consoleblank=0 earlycon¥0" ¥
→
"setup=setenv setupargs console=${console},${baudrate} consoleblank=0 earlycon vt.global_cursor_default=0¥0" ¥

修正内容に対するパッチを作成します。
[Ubuntu]$ quilt refresh

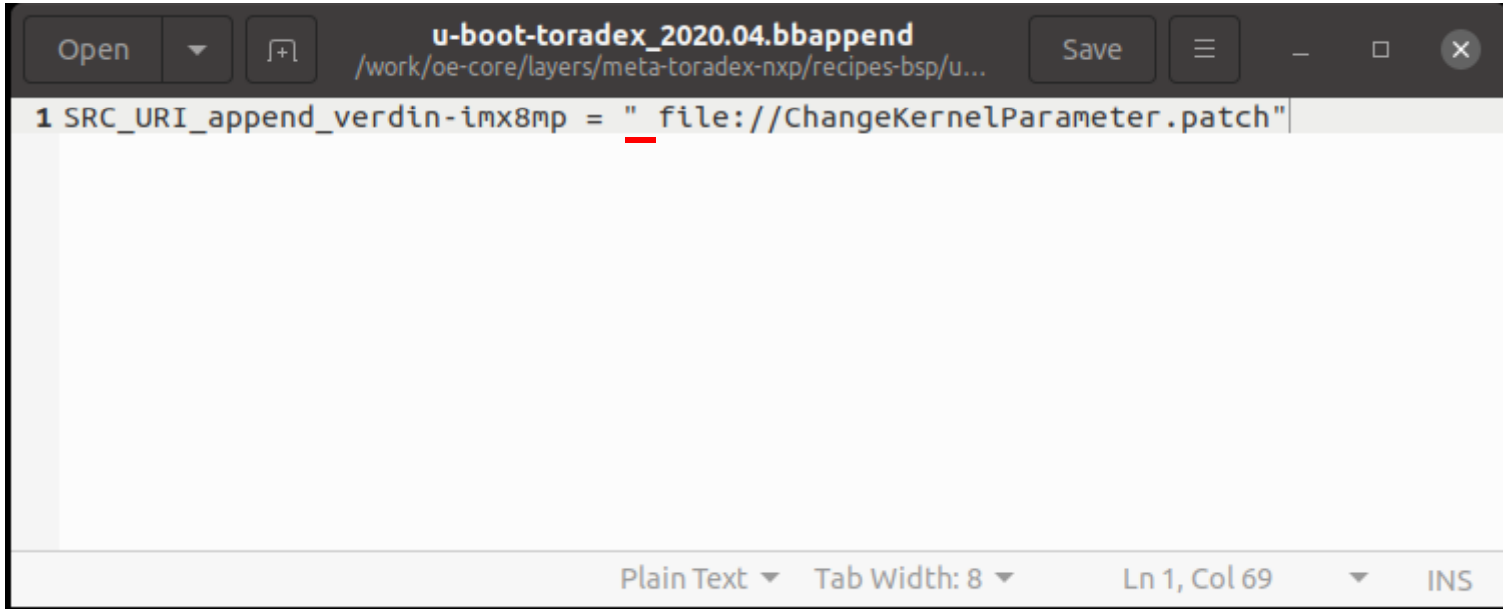
パッチ格納ディレクトリを作成します。
[Ubuntu]$ mkdir /work/oe-core//layers/meta-toradex-nxp/recipes-bsp/u-boot/files/verdin-imx8mp

パッチファイルを移動します。
[Ubuntu]$ mv ./patches/ChangeKernelParameter.patch /work/oe-core//layers/meta-toradex-nxp/recipes-bsp/u-boot/files/verdin-imx8mp/

devshellを終了します。
[Ubuntu]$ exit

bbappend**ファイルはないため作成します。**
[Ubuntu]$ gedit /work/oe-core/layers/meta-toradex-nxp/recipes-bsp/u-boot/u-boot-toradex_2020.04.bbappend

パッチをレシピに追加します。
内容は下記です。
SRC_URI_append_verdin-imx8mp = " file://ChangeKernelParameter.patch"



初期化(一度編集した内容を元に戻します。)

[Ubuntu]$ bitbake -c cleansstate virtual/bootloader

不要の場合もありますが一度OSイメージのcleansstateを実行します。

[Ubuntu]$ bitbake -c cleansstate tdx-reference-multimedia-image

イメージの作成

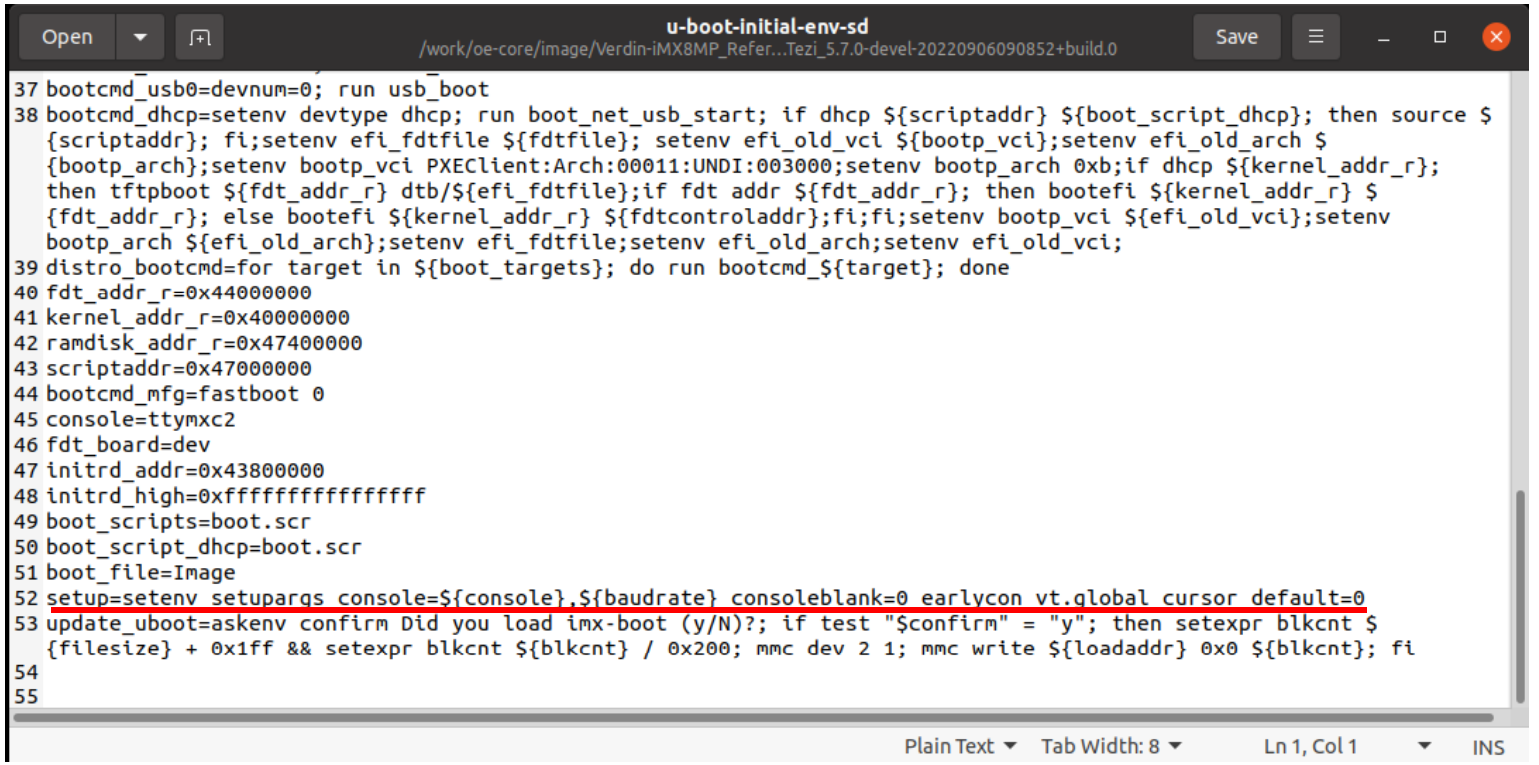
[Ubuntu]$ bitbake tdx-reference-multimedia-image

[Ubuntu]$ cd /work/oe-core/image/

解凍

[Ubuntu]$ sudo tar -xf ../build/deploy/images/verdin-imx8mp/Verdin-iMX8MP_Reference-Multimedia-Image-Tezi_<version>.tar

U-Bootの設定ファイルu-boot-initial-env-sdはU-Bootの修正が反映されて出力されます。



OSイメージを書き込んで起動するとLinux起動ロゴの下に起動ログが出力されなくなっていることが確認できます。