XG-335x
OpenCV アプリケーション開発

Rev1.0　2014/11/12

目　次

表記

●バージョンに関する表記

弊社提供のソース等に関しては、弊社の管理するバージョン番号がファイル名やフォルダ名に付いている場合があります。
そのバージョン番号に関しては、本ドキュメントでは、『**X**』を使用して表現しております。
そのため、以下のような表記になりますので、その部分は読み替えてください。

例：
以下の表記がある場合

helloworld-X.X.tar.bz2

Ver1.0 での実際のファイル名は、以下になります。

helloworld-1.0.tar.bz2

●コマンドラインの表記

本ドキュメントには、コマンドラインで入力する操作手順が記載されております。
操作は PC 及び XG ボードで行います。それぞれの記述について以下に記載します。

ゲスト OS(Ubuntu)での操作
プロンプトは、『**$**』で記載します。
実際のプロンプトには、カレントディレクトリ等が表示されますが、本ドキュメントでは省略します。
なお、省略時には、コマンドプロンプトの前に、 省略 と表記します。

XG ボード上の Linux での操作
プロンプトは、『**#**』で記載します。

本ドキュメント中での入力では、以下のように表現し、入力の最後には、 入力 があります。

例：ゲスト OS(Ubuntu)上で make コマンドを実行する場合の表記

```
省略 $ make  入力
```

コマンドによっては 1 つのコマンドが複数行で記載されている場合もあります。
その場合には、2 行目以降の入力では ENTER キーを押さずに続けて入力し、 入力 の表記がある行の最後で ENTER キーを入力してそのコマンドを実行してください。

例：2 行続いてコマンド入力がある表記

```
省略 $ mkimage -A arm -O linux -T ramdisk -C gzip -d output/images/rootfs.cpio.gz output/i
mages/uInitrd-xg3358  入力
```

1.　概要

1.1　はじめに

OpenCV は画像処理をする為のライブラリ集です。基本的な画像処理から、オブジェクト認識までいくつかのモジュールが用意されており、数百以上の関数が用意されています。
本ドキュメントでは、XG シリーズに OpenCV を使ったアプリケーションの動作環境を構築する方法を説明します。

本ドキュメントでは、XG シリーズの開発環境が Ubuntu12.04LTS 上にインストールされていることが前提となっています。開発環境をインストールされていない場合は、『**Linux 開発 インストールマニュアル**』に従って、先に開発環境の作成を行ってください。
また、XG-3358 では LCD-KIT に対応したカーネルも必要となります。カーネルの作成方法に関しては、各開発キットの『**ソフトウェアマニュアル**』で説明していますので、本ドキュメントを読む前にそちらのマニュアルも一通り行ってから本ドキュメントをお読みください。

1.2　対象機種（開発キット）

本アプリケーションノートは、以下の機種（開発キット）のバージョンを対象とします。

機種（開発キット）	バージョン
XG-3352(LK-3352-A01)	Rev1.1 以降
XG-3358(LK-3358-A01)	Rev1.1 以降
XG-BBEXT	Rev1.1 以降

バージョンは、付属 CD もしくは DVD のバージョン番号で表記しております。各ファイルの更新内容に関しては、機種（開発キット）の更新履歴でご確認ください。

1.3　OpenCV とは

OpenCV は画像処理ライブラリ集です。BSD ライセンスのオープンライブラリで、数百にも上るコンピュータヴィジョン（Computer Vision)アルゴリズムが含まれています。

2014/7 月時点での最新バージョンは 2.4.9 です。 2.4.x ではライブラリのインターフェースは C から C++への移行段階となっており、次期バージョンである 3.x では C++インターフェースのみになる予定です。

1.4　OpenCV の代表的なモジュール

OpenCV では沢山の機能があり、それらはモジュールという単位で機能ごとに分類されています。代表的なものには下表のようなモジュールがあります。

モジュール名	概要
core	基本的なデータ構造、Mat 構造体、共通で用いられる基本的な関数など
imgproc	イメージ処理モジュール
video	ビデオ解析モジュール
calib3d	基本的な多視点幾何アルゴリズム、シングルまたはステレオカメラ校正、物体の姿勢把握、ステレオ相応アルゴリズム、3D 再構築エレメント
feature2d	特徴点検出機能、記述機能、照合機能
objdetect	オブジェクトの検出
higui	便利な GUI インターフェース

1.5　XG シリーズ用 OpenCV を使ったアプリの開発

XG シリーズをターゲットとした OpenCV を使用したアプリのプログラム開発、実行(デバッグ）するには以下に挙げる準備が必要となります。

① XG シリーズで稼動させるためカスタマイズしたカーネルの作成（カメラを使用する場合など）
② アプリケーションを動かすための必要なライブラリをインストールしたルートファイルシステムの作成。
（Qt,OpenCV, gstreamer、OpenSSH など各種パッケージが必要となります）
③ Ubuntu 上でクロス開発するための開発環境のインストール、設定（Qt、Eclipse など）

ここでは XG シリーズにて OpenCV を使ったアプリケーションの開発、実行に必要となる項目について説明します。Qt の詳細についてはアプリケーションマニュアル「Qt のためのルートファイルシステム作成方法」および「Qt(4.8.5)プログラミング」を参照してください。

2.　OpenCV アプリの実行環境概要

2.1　システム概要

組込みシステムにおいて OpenCV を使用したアプリには 3 通りのタイプがあります。
1 つ目は、画像変換など画像データ処理のみで画像キャプチャ入力／グラフィック表示を必要としないものです。たとえば Web の CGI プログラムなどが該当します。
2 つ目は、カメラなどを使用して画像をキャプチャし何らかの処理をするものです。
3 つ目は、何らかの処理した画像をグラフィック表示するものです。(タッチパネルによる GUI 操作も含みます)
2 と 3 のタイプは組み合わせて使用することもよくあります。

XG シリーズでは、これらのタイプに対して以下のように対応が可能です。

機種	画像変換などの処理	キャプチャ入力	GUI 操作
XG-3358	ルートファイルシステムの再構築	カーネル、ルートファイルシステムの再構築	カーネル、ルートファイルシステムの再構築（LCD キットが必要です）
XG-BBEXT			ルートファイルシステムの再構築
XG-3352			不可

OpenCV アプリケーションのクロス開発環境を整えるには、Ubuntu 側にてルートファイルシステムの再構築が必要となります。

2.2　OpenCV システム構成

①　カメラ、LCD 表示のない場合

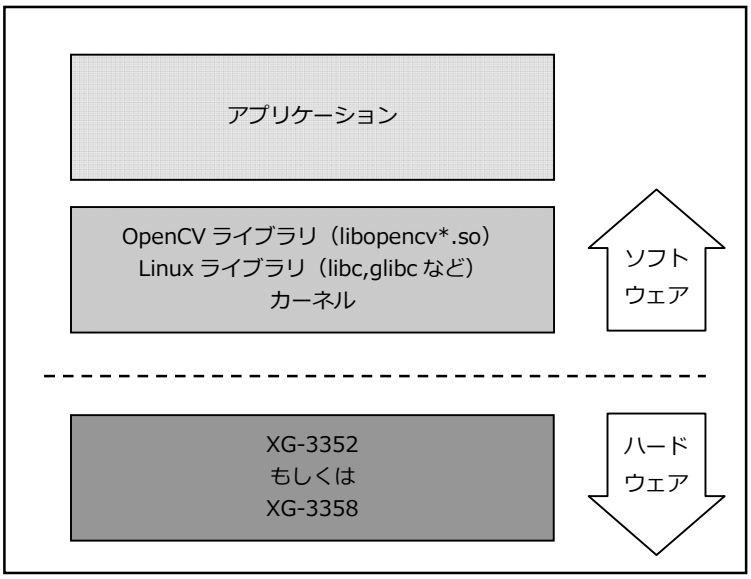


Fig 2.2-1　OpenCV アプリ動作環境（カメラ、LCD 表示なし）

②　カメラのみの場合

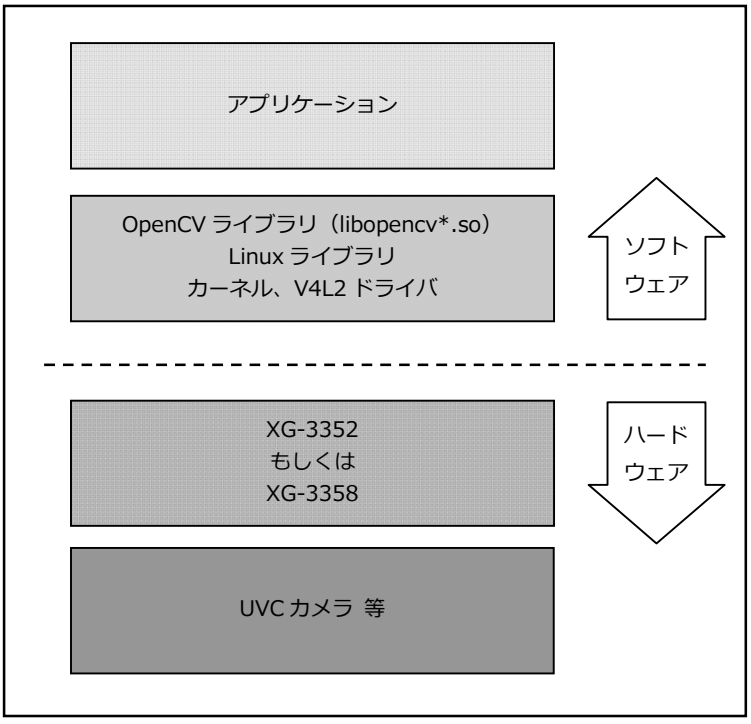


Fig 2.2-2　OpenCV アプリ動作環境（カメラのみ、LCD 表示なし）

③　LCD 表示をする場合

GUI 操作を伴う OpenCV アプリケーションを実行する場合は Qt などの GUI ツールを使用します。Qt のほかにも GTK2+などもありますが此処では Qt を使用した例を紹介します。Qt を動かすための構成は幾つかありますが、ここでは一番コンパクトでオーバヘッドの少ない framebuffer をバックエンドにした構成を例にとって説明します。

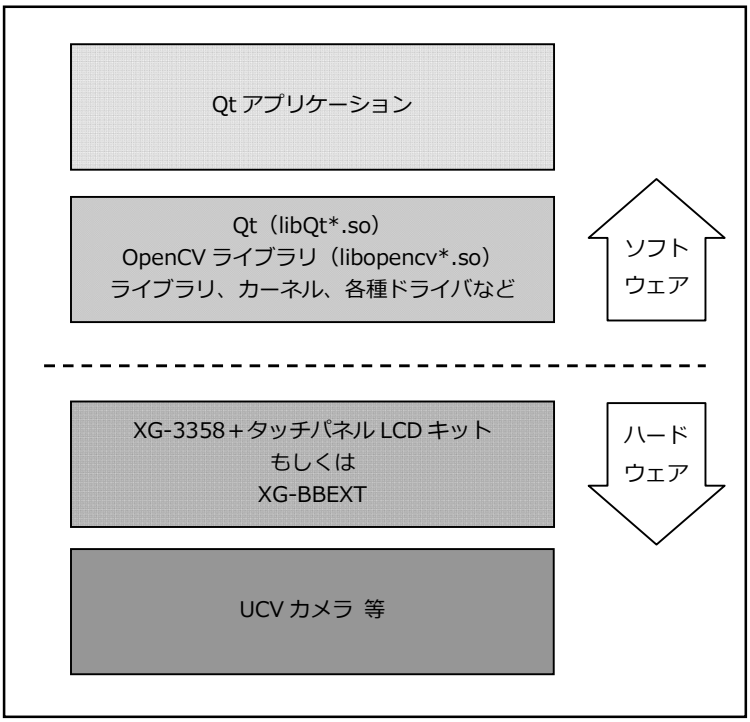


Fig 2.2-3 OpenCV アプリ動作環境（LCD 表示/GUI 操作）

2.3　OpenCV サンプルアプリケーション

サンプルとして 2 種類の OpenCV アプリケーションの開発から実行までの手順を紹介します。ルートファイルシステムのビルド、OpenCV ライブラリを使ったアプリケーションの作成、および実行方法を以下の 3 章、4 章にて解説します。

① 3 章：サンプル 1（LCD の無い XG-3352, XG-3358 でも実行が可能です）
ターミナルからコマンド引数に画像ファイルを指定すると、2 値化してファイル保存します。
NFS 上に保存すれば、Ubuntu 側で Nautilus を使ってそのまま結果を確認することができます。

Fig 2.3-1 変換元画像

Fig 2.3-2 2 値化変換画像

② 4 章：サンプル 2（Qt の GUI 機能を使ったサンプルで XG-3358(LCD キット）または XG-BBEXT 用です）
ターミナルからコマンド引数に画像ファイルを指定し起動します。起動時は指定された画像が表示されます。
タッチパネルの「エッジ検出」ボタンをタッチすると 2 値化した画像が表示されます。「閉じる」ボタンをタッチするとプログラムは終了します。
開発環境としては Qt Creator を使用します。

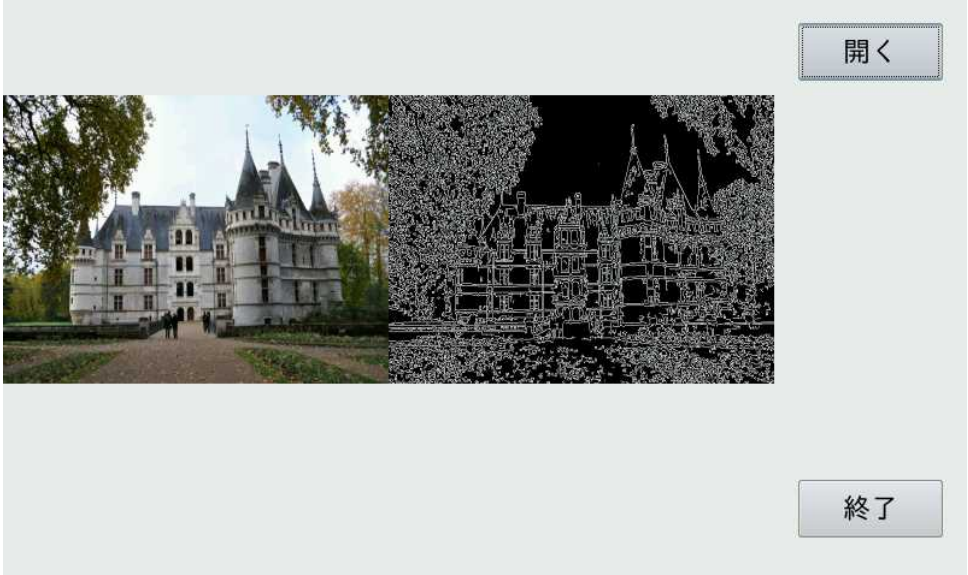


Fig 2.3-3 Qt によるエッジ検出 LCD 画面

3.　サンプル１　(2 値化)

OpenCV を使ったアプリケーションの場合、キャプチャ入力(カメラ)、グラフィック表示(LCD)などが伴うことが多いのですが、組込み用途や Web サーバなどでは画像データ変換のみで、これらハードウェアを実装しないこともあります。このような場合 XG シリーズの標準カーネルのままで OpenCV アプリケーションを動かすことができます。

ルートファイルシステムのビルドおよび XG ボードへのインストールなどの詳細な手順については XG シリーズ各ソフトウェアマニュアルを参考にしてビルドしてください。

ここでは XG-3352 を例にとって説明しますが、XG-3358, XG-BBEXT も同じ手順です、ディレクトリのパスを各 XG ボードに読み替えてください。

3.1　ルートファイルシステムのビルド

OpenCV 関連のパッケージをインストールします。

ルートファイルシステム作成の準備

①　ビルドルートシステムのディレクトリに移動します

```
省略 $ cd ~/xg3352-lk/buildroot-2013.11-xg3352-X.X      入力
```

上記は、LK-3352-A01 の時のディレクトリ指定となります。移動先ディレクトリは機種（開発キット）ごとに異なりますので適宜変更してください。
なお、以下の表に各機種（開発キット）ごとのディレクトリ名を記載しますが、バージョンによって異なる場合がありますので、各機種（開発キット）のソフトウェアマニュアルを参照ください。

機種（開発キット）	ディレクトリ
XG-3352(LK-3352-A01)	~/xg3352-lk/buildroot-2013.08.1-xg3352-X.X
XG-3358(LK-3358-A01)	~/xg3358-lk/buildroot-2013.11-xg3358-X.X
XG-BBEXT	~/xgbbext-lk/buildroot-2013.11-xgbbext-X.X

②　output/target ディレクトリ下に配置したファイル、および変更したファイルがありましたら、必要に応じてバックアップを取ります。

③　ルートファイルシステムを再構築するため、一度ビルドしてある output 下のディレクトリ、ファイルを以下のコマンドで削除します。

```
省略 $ make distclean      入力
```

「make distcclean」および 「make clean」を実行すると、output ディレクトリ内のディレクトリ、ファイルが削除されます。

④　XG シリーズの buildroot のコンフィグレーションを行います。

```
省略 $ make xg3352_defconfig    入力
```

上記は、LK-3352-A01 の時のコマンドとなります。コマンドは機種（開発キット）ごとに異なりますので適宜変更してください。
なお、以下の表に各機種（開発キット）ごとのコンフィグレーション名を記載しますが、バージョンによって異なる場合がありますので、各機種（開発キット）のソフトウェアマニュアルを参照ください。

機種（開発キット）	コンフィグレーション名
XG-3352(LK-3352-A01)	xg3352_defconfig
XG-3358(LK-3358-A01)	xg3358_defconfig
XG-BBEXT	xgbbext_defconfig

「make distclean」は「make clean」に加えて、Makefile, config.cache などのファイルも削除されます。Makefile のコンフィグレーション設定を変更した場合は「make distclean」を実行する必要があります。

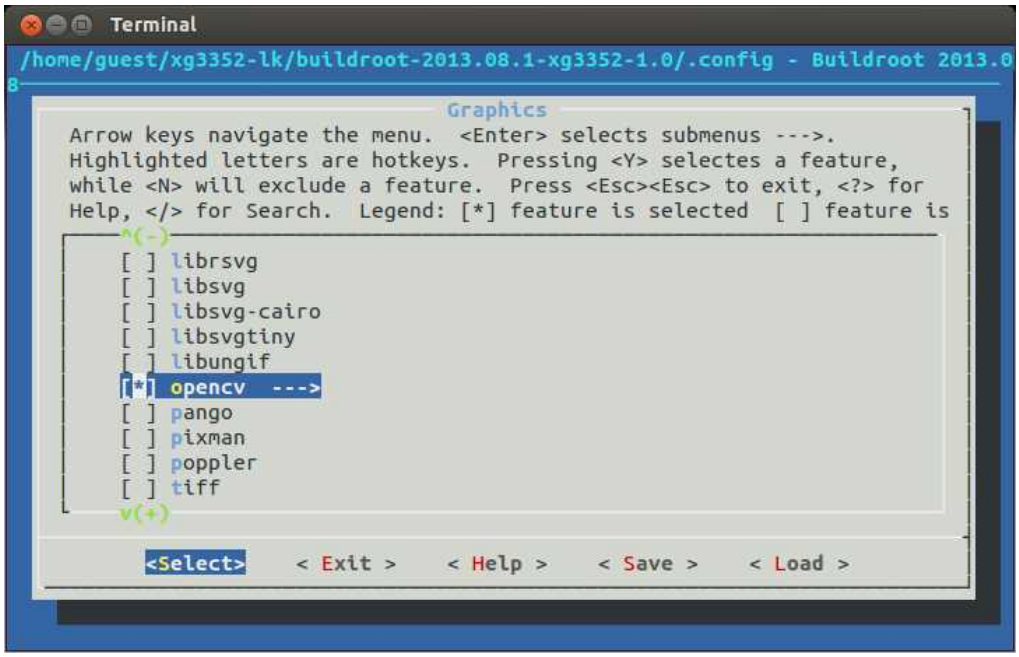
ルートファイルシステムのカスタマイズ

① ルートシステムのカスタマイズのためのメニューを起動します。

```
省略 $ cd ~/xg3352-lk/buildroot-2013.08.1-xg3352-X.X   入力
省略 $ make menuconfig   入力
```

② OpenCV を選択します

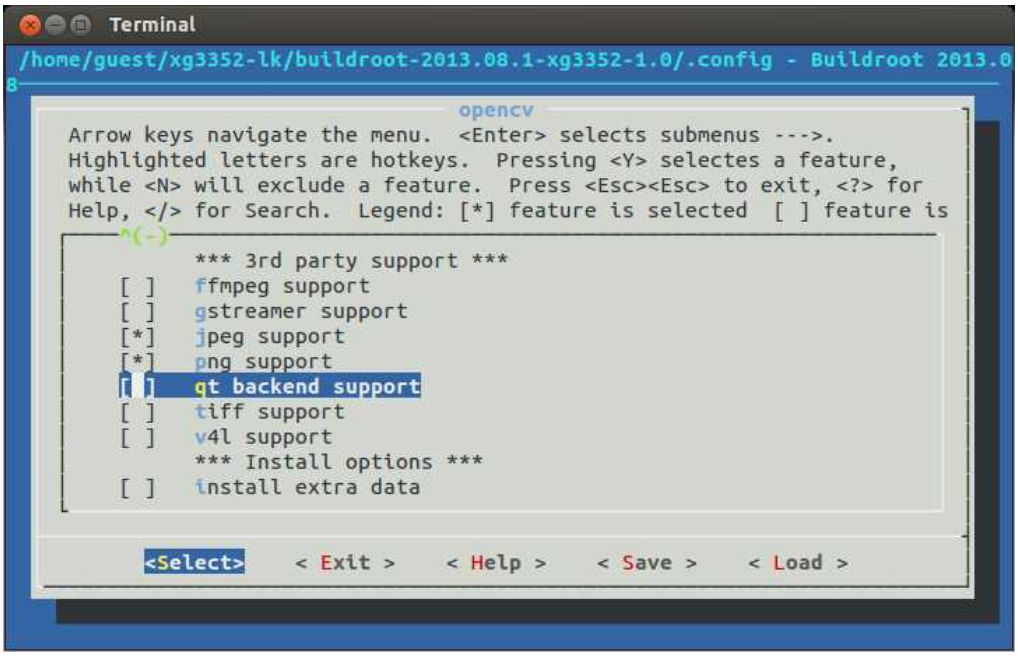
「Target packages」－「Libraries」－「Graphics」メニューを開いて「OpenCV」を選択します。OpenCV に関する詳細な設定ができるようになります。Enter キーを押して OpenCV の設定メニューに移動します。



③ OpenCV のモジュール、サードパーティのサポートなどを選択します。

画像ファイルを扱うので「jpeg support」と「png support」を選択します。

Qt は使用しないのでデフォルトの設定から「qt backent support」の選択を外します。



④　一通りの設定が終了したら ESC-ESC を数回押して保存問い合わせが
出るまで戻り、Yes を選択して設定内容を保存して menuconfig を終了します。

ルートファイルシステムのビルド

①　Buildroot のディレクトリに移動します。

```
省略 $ cd ~/xg3352-lk/buildroot-2013.08.1-xg3352-X.X    入力
```

②　ルートファイルシステムのイメージを作成するため、再びビルドします。

```
省略 $ make    入力
```

config の内容によっては、必要なパッケージのソフトをダウンロードすることになりますので、ネットワークに接続した環境にて make を行います。
make の途中でエラーが発生することがあります。エラーメッセージを参考にエラー原因を取り除いて再度 make します。よくある原因としては、以下の内容があります。

- 開発 PC に必要なパッケージ・ライブラリがインストールされていない。
- 開発 PC の Linux、バージョンが適応しない。（Ubuntu14.04 など）
- Toolchain のバグ（最適化レベルを変えてみたりして回避します）
- ダウンロードサイトがメンテナンス中である。
- menuconfig で選択したパッケージが足りない。

作成されたルートファイルシステムのインストールについては各 XG シリーズのソフトウェアマニュアルを参考にして下さい。

3.2　アプリケーションの作成

ここでは IDE(統合開発環境）を使わずにアプリケーションを開発する手順を説明します。

ソースプログラムの作成

ソースプログラムを作成します。vi あるいは gedit で以下のソースコードを入力してください。
ソースファイル名は「cv_sample1.cpp」とします。

```
省略 $ mkdir -p ~/opencv_test/sample1    入力
省略 $ cd ~/opencv_test/sample1          入力
省略 $ gedit cv_sample1.cpp              入力
```

[cv_sampel1.cpp]

```
#include <opencv2/opencv.hpp>

using namespace cv;

int main(int argc, char **argv)
{
  if( argc != 3 ) {
    std::cerr << "使用方法: cv_sample1 入力画像ファイル名　変換出力ファイル名¥n";
    return -1;
  }
  // イメージファイルを読み込みグレイスケールに変換
  Mat src_img = imread( argv[1] );
  if(!src_img.data) {
    std::cerr << "画像ファイルの読込みができませんでした¥n";
    return -1;
  }
  Mat gray_img;
  cvtColor(src_img, gray_img, CV_BGR2GRAY);
  // 固定閾値処理
  Mat bin_img;
  threshold(gray_img, bin_img, 0, 255, THRESH_BINARY|THRESH_OTSU);
  // ファイルに出力
  imwrite( argv[2], bin_img );
  return 0;
}
```

ビルド（コンパイル）

cv_sample1 は、C++のソースコードは 1 ファイルのみなので make, cmake を使わずに、そのままコマンドラインで実行します。
1 番目のコマンドの環境変数「BR_ROOT」の設定は、ターミナル（の各ページ）を開いたときに一度だけ実行してください。

```
省略 $ export BR_ROOT=$HOME/xg3352-lk/buildroot-2013.08.1-xg3352-1.0   入力
省略 $ $BR_ROOT/output/host/usr/bin/arm-linux-gnueabihf-gcc -Wl,--sysroot=$BR_ROOT/outp
ut/host/usr/arm-buildroot-linux-gnueabihf/sysroot `pkg-config --cflags opencv` -o cv_sa
mple1 cv_sample1.cpp -lopencv_core -lstdc++ -lopencv_highgui -lopencv_imgproc    入力
省略 $ ls    入力
cv_sample1  cv_sample1.cpp  cv_sample1.cpp~
```

「pkg-config --libs opencv」は使用せずに、--sysroot によるルートパス指定と、-l オプションによる個別ライブラリ指定をしてください。

「pkg-config --libs opencv」を使用すると、ライブラリパスが絶対パスで返されるため、Ubuntu 側の OpenCV ライブラリとリンクしようとしてリンクエラーが発生します。

出来上がった実行可能ファイルは NFS 共有ディレクトリにコピーするか、XG ボードにコピーして使用します。
ここでは NFS 共有ディレクトリにコピーします。

```
省略 $ cp cv_sample1 /nfs/    入力
```

テスト用に画像ファイルも何枚かコピーします。

アプリケーションノートで使用しているサンプル画像は以下のサイトよりダウンロードできます。
「Creative Commons license」で提供されております。

http://www.freephotobank.org/main.php

3.3　実行

実行

root でログインします。パスワード設定してあるときはパスワードも入力してください。

```
Welcome to Buildroot
xg-3352 login: root    入力
Password:

# mount -t nfs -o nolock 192.168.128.210:/nfs /mnt/nfs    入力
# cd /mnt/nfs    入力
# ./cv_sample1 Azy-le-Rideau-05.jpg test.png    入力
```

Azy-le-Rideau-05.jpg(元画像)

test.png（2 値化変換画像）

4.　サンプル２(エッジ検出)

OpenCV ライブラリを使用したアプリケーションを動かすためのルートファイルシステムをビルドするには buildroot の make menuconfig において OpenCV を指定します。さらに LCD 表示/GUI 操作をするに此処では Qt もビルドするようにします。
Qt のインストールについての詳細はアプリケーションノート「Qt のためのルートファイルシステム作成方法」を参考にしてください。

ルートファイルシステムのスケルトンは、ルートファイルシステムをビルドするときのルートファイルシステムの構造の基礎となる部分です。この部分に Qt で必要となるファイル、フォントをあらかじめセットしておきます。(microSD などの書き換え可能なデバイス上にルートファイルシステムを構築する場合は、buildroot でファイルシステムを構築後、書き換えることも可能です。)

4.1　ルートファイルシステムのビルド

ルートファイルシステム作成の準備

① ビルドルートシステムのディレクトリに移動します

```
省略 $ cd ~/xg3358-lk/buildroot-2013.11-xg3358-X.X    入力
```

上記は、LK-3358-A01 の時のディレクトリ指定となります。移動先ディレクトリは機種（開発キット）ごとに異なりますので適宜変更してください。
なお、以下の表に各機種（開発キット）ごとのディレクトリ名を記載しますが、バージョンによって異なる場合がありますので、各機種（開発キット）のソフトウェアマニュアルを参照ください。

機種（開発キット）	ディレクトリ
XG-3358(LK-3358-A01)	~/xg3358-lk/buildroot-2013.11-xg3358-X.X
XG-BBEXT	~/xgbbext-lk/buildroot-2013.11-xgbbext-X.X

② output/target ディレクトリ下に配置したファイル、および変更したファイルがありましたら、必要に応じてバックアップを取ります。

すでに Qt が稼動するルートファイルシステムがビルド済みの場合は、手順③～④は省いてルートファイルシステムのカスタマイズに進んでください。

③ ルートファイルシステムを再構築するため、一度ビルドしてある output 下のディレクトリ、ファイルを以下のコマンドで削除します。

```
省略 $ make distclean    入力
```

「make distcclean」および 「make clean」を実行すると、output ディレクトリ内のディレクトリ、ファイルが削除されます。

④　XG シリーズの buildroot のコンフィグレーションを行います。

```
省略 $ make xg3358_defconfig    入力
```

上記は、LK-3358-A01 の時のコマンドとなります。コマンドは機種（開発キット）ごとに異なりますので適宜変更してください。
なお、以下の表に各機種（開発キット）ごとのコンフィグレーション名を記載しますが、バージョンによって異なる場合がありますので、各機種（開発キット）のソフトウェアマニュアルを参照ください。

機種（開発キット）	コンフィグレーション名
XG-3358(LK-3358-A01)	xg3358_defconfig
XG-BBEXT	xgbbext_defconfig

「make distclean」は「make clean」に加えて、Makefile, config.cache などのファイルも削除されます。Makefile のコンフィグレーション設定を変更した場合は「make distclean」を実行する必要があります。

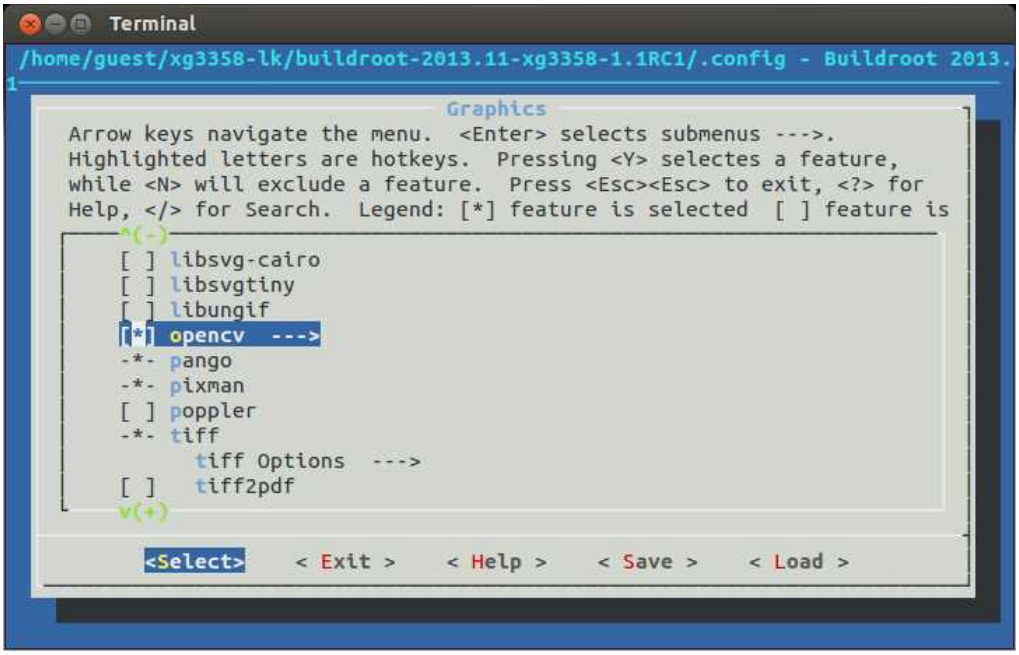
ルートファイルシステムのカスタマイズ

① ルートシステムのカスタマイズのためのメニューを起動します。

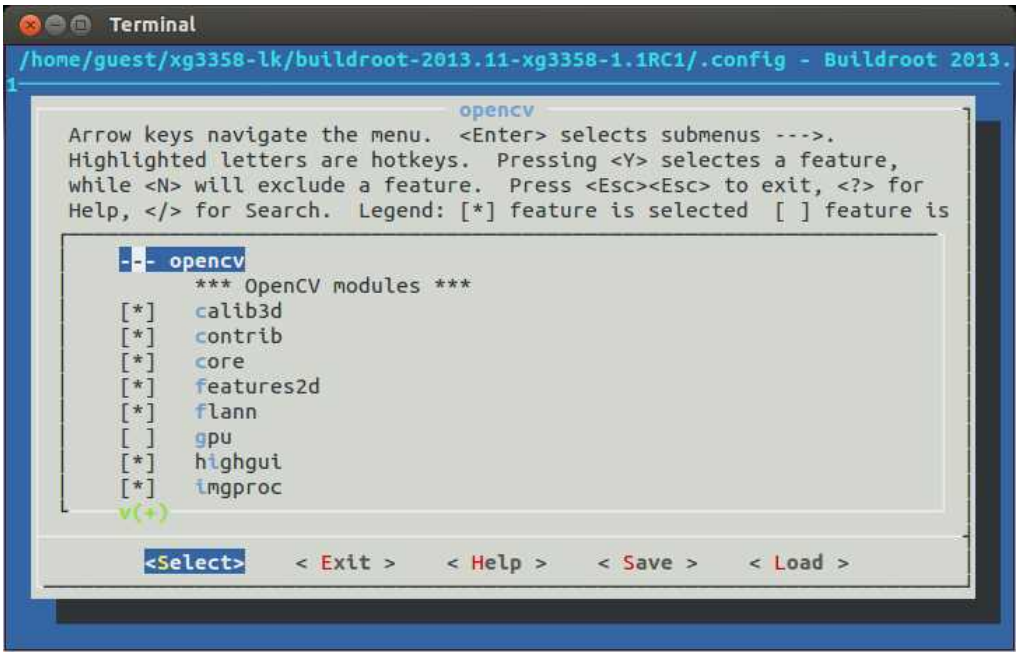
```
省略 $ cd ~/xg3358-lk/buildroot-2013.11-xg3358-X.X   入力
省略 $ make menuconfig   入力
```

② 「Qt アプリケーションノート」を参考にして Qt に関する設定または設定の確認をします。

③ OpenCV の設定をします。
「Target packages」－「Libraries」－「Graphics」－「opencv」を選択します。Enter キーを押すと詳細設定に移ります。



④ OpenCV の詳細設定で必要なモジュール、バックエンド、機能を選択します。



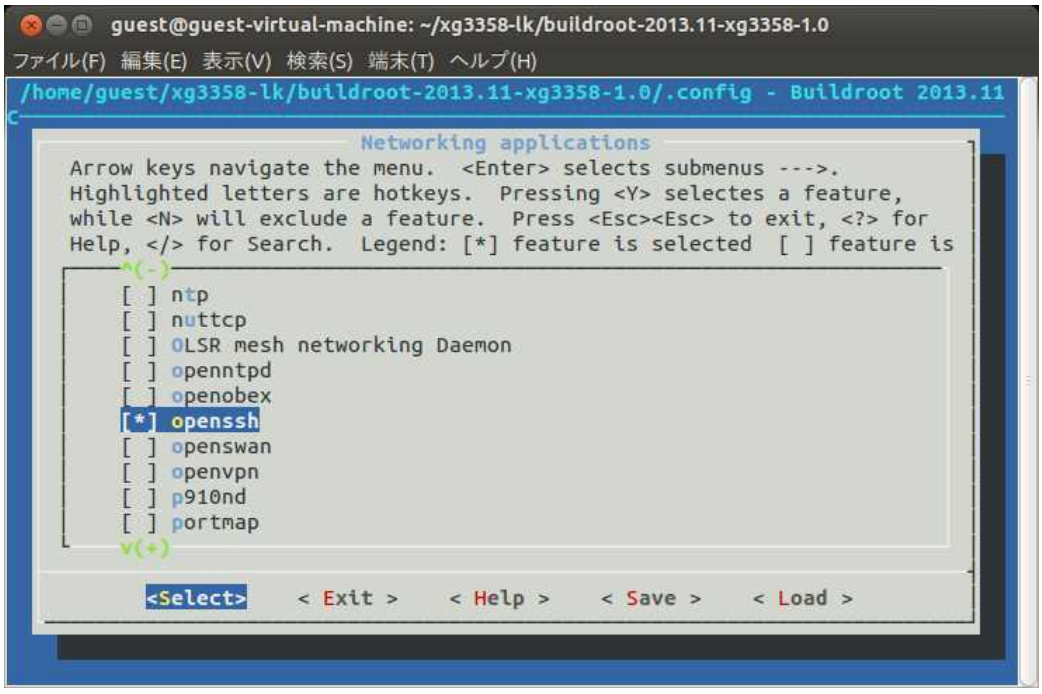
選択項目について。どのモジュールを必要とするかは開発する OpenCV を使ったアプリケーションプログラムによります。どれを選択してよいか分からない場合は、ほとんどのモジュール、バックエンド、機能を選択します。アプリケーションが完成した後、必要に応じて不要なモジュールをカットします。
ここで選択する必要がないモジュールとして「gpu」があります。現時点では AM335X 系の PowerVR/SGX はサポートされていません。
また nonfree モジュールは一部の国によっては特許権が成立していたり、使用において制限があるなど法的に問題となるアルゴリズムを含んでいます。nonfree モジュールを使用するにあたってはご注意ください。
（nonfree には現在 SIFT, SURF が含まれています）

⑤ そのほかのパッケージ、ライブラリ、モジュールの設定。
OpenCV の各モジュールを選択すると、必要となる最低限の関連したパッケージ、ライブラリのビルド設定がされます。しかしながら実際には必要とするものが足りないなどでビルドエラーが発生することもありますが、このような場合はエラーメッセージを参考に必要となるパッケージ、ライブラリを探してビルド対象に加え再度ビルドします。

ここではよく必要とされるパッケージ、ライブラリなどをまとめてみました。特別インストールをしない理由がなければインストールした方が無難です。（XG シリーズ、Qt やマルチメディアパッケージの設定によっては必須あるいは、すでにインストール設定されているものもあります。）

設定項目	内容
gstreamer	動画確認などに使用します
ffpmeg	画像変換やストリーム配信などに使用します
orc	データ配列の処理に内部的に使用します

⑥ OpenSSH を有効にします。「Target packages」－「Newtworking applications」メニューを開いて「openssh」を選択します。（Qt のリモートデバッグに必要です。デバッグに関してはアプリケーションノート「Qt4.8.5 プログラミング」を参照して下さい。



⑦ 一通りの設定が終了したら ESC-ESC を何度か押して保存問い合わせが出るまで戻り、Yes を選択して設定内容を保存して menuconfig を終了します。

ルートファイルシステムのビルド（パッケージのビルド）

① Buildroot のディレクトリに移動します。

```
省略 $ cd ~/xg3358-lk/buildroot-2013.11-xg3358-X.X
```
入力

② ルートファイルシステムのイメージを作成するためパッケージのビルドします。
Qt、OpenCV、gstreamer などの選択した項目によってはビルドに数時間程度かかることもあります。

```
省略 $ make
```
入力

config の内容によっては、必要なパッケージのソフトをダウンロードすることになりますので、ネットワークに接続した環境にて make を行います。
make の途中でエラーが発生することがあります。エラーメッセージを参考にエラー原因を取り除いて再度 make します。よくある原因としては、以下の内容があります。

- 開発 PC に必要なパッケージ・ライブラリがインストールされていない。
- 開発 PC の Linux、バージョンが適応しない。（Ubuntu14.04 など）
- Toolchain のバグ（最適化レベルを変えてみたりして回避します）
- ダウンロードサイトがメンテナンス中である。
- menuconfig で選択したパッケージが足りない。

各種設定ファイルの編集

① output/target フォルダに移動します。

```
省略 $ cd ~/xg3358-lk/buildroot-2013.11-xg3358-X.X/output/target  入力
```

② etc/profile へ環境変数を追加します。(XG-BBEXT の場合は設定済みです)

Qt アプリケーションで必要となる環境変数は下図のとおりです。バックエンドによって必用な環境は異なります。

XG-3358 の場合

環境変数名	設定内容
LANG	'ja_JP.UTF-8'
TZ	JST-9
TSLIB_CONSOLEDEVICE	none
TSLIB_TSDEVICE	/dev/input/event0
QWS_MOUSE_PROTO	'tslib:/dev/input/event0'
QWS_KEYBOARD	"LinuxInput:/dev/input/event0"
QWS_DISPLAY	"LinuxFB:mmWidth=225:mmHeight=135"

XG-BBEXT の場合

環境変数名	設定内容
LANG	'ja_JP.UTF-8'
TZ	JST-9
TSLIB_CONSOLEDEVICE	none
TSLIB_TSDEVICE	/dev/input/event1
QWS_MOUSE_PROTO	'tslib:/dev/input/event1'
QWS_KEYBOARD	"LinuxInput:/dev/input/event0"
QWS_DISPLAY	"LinuxFB:mmWidth=225:mmHeight=135" POWERVR 使用時は以下の設定にします "powervr:mmWidth=225:mmHeight=135"

```
省略 $ vi etc/profile  入力
```

以下のテキストを「etc/profile」の環境変数定義の最後に追加します。

[XG-3358]

```
export LANG='ja_JP.UTF-8'
export TZ=JST-9
export TSLIB_CONSOLEDEVICE=none
export TSLIB_TSDEVICE=/dev/input/event0
export QWS_MOUSE_PROTO='tslib:/dev/input/event0'
export QWS_KEYBOARD="LinuxInput:/dev/input/event0"
export QWS_DISPLAY=LinuxFB:mmWidth=225:mmHeight=135
```

[XG-BBEXT]

```
export LANG='ja_JP.UTF-8'
export TZ=JST-9
export TSLIB_CONSOLEDEVICE=none
export TSLIB_TSDEVICE=/dev/input/event1
export QWS_MOUSE_PROTO='tslib:/dev/input/event1'
export QWS_KEYBOARD="LinuxInput:/dev/input/event0"
export QWS_DISPLAY=LinuxFB:mmWidth=225:mmHeight=135
```

PowerVR を使用するときは「QWS_DISPLAY」の指定は powervr にしてください。

③　必要に応じてネットワーク設定します。

```
省略 $ vi etc/network/interfaces  入力
```

④　必要に応じてフォントファイルのインストールをします。
フォントファイルのダウンロード、インストールの手順はアプリケーションマニュアル「Qt のためのルートファイルシステム作成方法」を参照してください。

output/target にコピーしたファイルや output/target ディレクトリ内の修正したファイルは「make clean」などを行いますと消去されます。system/skeleton-xg3358 の相当するディレクトリに、必要なファイルをコピーしておくと「make clean」後、ビルドすると skeleton-xg3358 からルートファイルのスケルトンイメージがコピーされます。恒久的に追加したいファイルなどは skeleton-xg3358 の相当するディレクトリにコピーしておくと便利です。

skeleton のパス

機種（開発キット）	ディレクトリ
XG-3358(LK-3358-A01)	~/xg3358-lk/buildroot-2013.11-xg3358-X.X/system/skeleton-xg3358
XG-BBEXT	~/xgbbext-lk/buildroot-2013.11-xgbbext-X.X/system/skeleton-xg-bbext

libQtTtest.so.*ファイルのコピー

OpenCV で必要となる「libQtTest.so.*」は Qt をビルドしても自動的にターゲットにはインストールされません。

以下の手順で「libQtTest.so.*」をコピーします。

①　「libQtTetst.so.*」ファイルのあるディレクトリに移動します

```
省略 $ cd ~/xg3358-lk/buildroot-2013.11-xg3358-X.X/output/build/qt-4.8.5/lib  入力
```

②　ファイルをコピーします。

```
省略 $ cp -a libQtTest.so.* ../../../target/usr/lib  入力
```

ルートファイルシステムのビルド（ルートファイルシステムイメージの作成）

① Buildroot のディレクトリに移動します。

```
省略 $ cd ~/xg3358-lk/buildroot-2013.11-xg3358-X.X        入力
```

② ルートファイルシステムのイメージを作成するため、再びビルドします。

```
省略 $ make        入力
```

作成されたルートファイルシステムのインストールについては、各 XG シリーズのソフトウェアマニュアルを参考にして下さい。

4.2　ターゲットボードの動作確認

環境変数の確認

ビルドしたルートファイルシステムが Qt に必要な環境で設定されているかどうか確認します。root でログインし、printenv で環境変数がセットされているかどうか確認します。

[XG-3358 の例]

```
xg-3358 login: root   入力
# printenv    入力
HISTFILESIZE=1000
INPUTRC=/etc/inputrc
TSLIB_TSDEVICE=/dev/input/event0          タッチパネルデバイスを指定
USER=root
HOSTNAME=xg-3358
HOME=/root
PAGER=/bin/more
PS1=#
TSLIB_CONSOLEDEVICE=none
LOGNAME=root
TERM=vt100
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/bin/X11:/usr/local/bin
LANG=ja_JP.UTF-8          言語は日本語、コードは UTF-8
DMALLOC_OPTIONS=debug=0x34f47d83,inter=100,log=logfile
HISTSIZE=1000
SHELL=/bin/sh
QWS_DISPLAY=LinuxFB:mmWidth=225:mmHeight=135      Qt Display タイプ、LCD サイズを指定
PWD=/root
TZ=JST-9          タイムゾーンは日本時間に
QWS_MOUSE_PROTO=tslib:/dev/input/event0      Qt マウス（タッチパネル）を指定
EDITOR=/bin/vi
```

[XG-BBEXT の例]

```
xg-bbext login: root    入力
# printenv    入力
HISTFILESIZE=1000
INPUTRC=/etc/inputrc
TSLIB_TSDEVICE=/dev/input/event1      タッチパネルデバイスを指定
USER=root
HOSTNAME=xg-bbext
HOME=/root
PAGER=/bin/more
PS1=#
QWS_KEYBOARD=LinuxInput:/dev/input/event0      タッチボタン デバイスを指定
TSLIB_CONSOLEDEVICE=none
LOGNAME=root
TERM=vt100
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/bin/X11:/usr/local/bin
LANG=ja_JP.UTF-8          言語は日本語、コードは UTF-8
DMALLOC_OPTIONS=debug=0x34f47d83,inter=100,log=logfile
HISTSIZE=1000
SHELL=/bin/sh
QWS_DISPLAY=powervr:mmWidth=225:mmHeight=135      Qt Display タイプ、LCD サイズを指定
PWD=/root
TZ=JST-9        タイムゾーンは日本時間に
QWS_MOUSE_PROTO=tslib:/dev/input/event1      Qt マウス（タッチパネル）を指定
EDITOR=/bin/vi
```

タッチパネル キャリブレーション

次にタッチパネルのキャリブレーションを行います。

```
# ts_calibrate
```

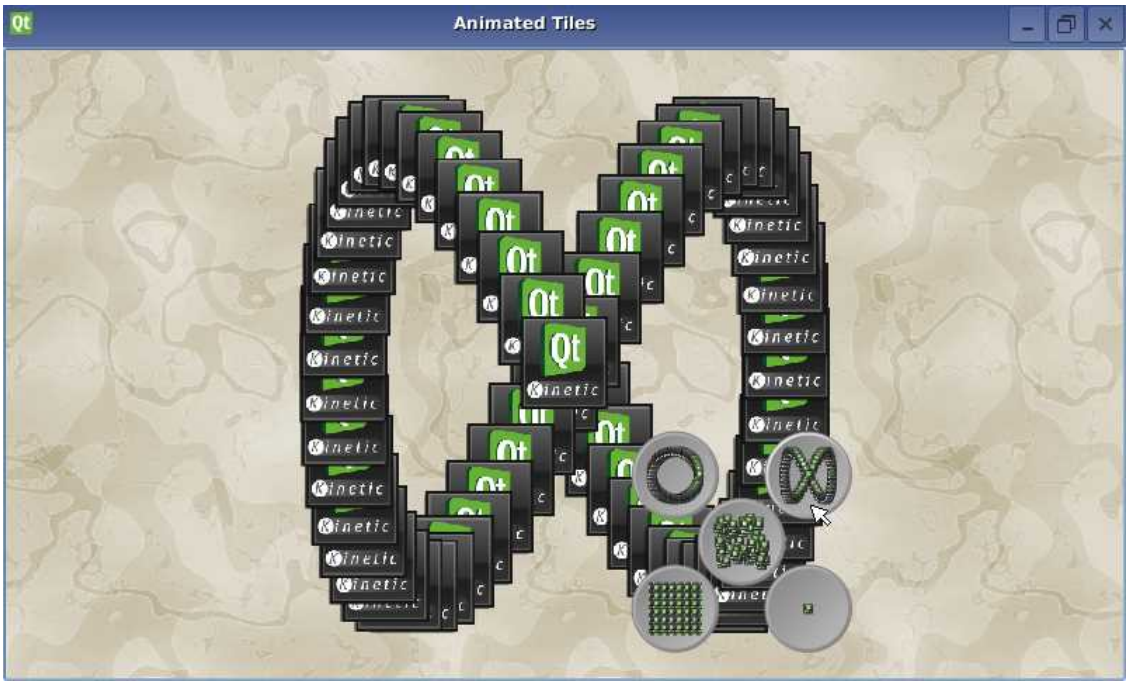
「+」が表示されますので、順にその位置をタッチしてください。タッチパネルのキャリブレーション情報は「/etc/pointercal」に格納されます。このファイルが無いと Qt アプリケーションを動かすことができませんので、必ずキャリブレーションをしてください。

Qt の動作確認

Buildroot の menuconfig にて、Qt デモおよび例をインストールした場合は Qt の動作確認が出来ます。
これらデモおよび例は、/usr/share/qt に格納されています。ただし、組み込み用の Qt embedded 用として特別に用意されているデモおよび例ではありませんので、組み込みシステムでは利用できないものもたくさん含まれています。

① 画面表示とタッチパネルの動作確認をします。
ここではタイルのアニメーション例で動作確認します。

```
# cd /usr/share/qt/examples/animation/animatedtiles    入力
# ./animatedtiles -qws    入力
```



画面上の 5 つのボタンを押すと、Qt タイルが移動しボタンのイメージの形に整列します。

デモ、例はソースコードも一緒にインストールされていますので Qt プログラミングの参考にしてください。

Qt アプリケーションを動かす場合、Qt Embedded Linux Server が必要となります。オプション「-qws」と付けることによって、プログラムをサーバとして起動することが出来ます。

サーバとして起動している Qt アプリケーションがすでに存在してる場合、別の Qt アプリケーションを起動するときには「-qws」オプションは必要ありません。サーバとして起動してる Qt アプリケーションを先に閉じると、そのほかの Qt アプリケーションもエラーで終了します。

後で起動した Qt アプリケーションに「-qws」オプションが付加されていると GUI イベントを取り合う形になり、操作が正常に出来なくなります。

ssh の動作確認

リモート PC から XG-3358、XG-BBEXT を操作するには、シリアルあるいはネットワーク接続が必要になります。統合開発環境下でデバッグする場合通常ターゲットボードとの接続はセキュリティの関係上 ssh が用いられます。「4.3」ルートファイルシステムのビルド」にて OpenSSH を有効にしてルートファイルシステムをビルドすれば ssh をはじめとするコマンドが利用できるようになります。ssh 関連のコマンドではターゲットボートとの通信は暗号化されます。Linux の最初の起動時に RSA の鍵の生成が行われます。

[RSA 鍵が生成されたときのログの例]

```
Generating RSA Key...
/usr/bin/ssh-keygen: /usr/lib/libcrypto.so.1.0.0: no version information
available (required by /usr/bin/ssh-keygen)
Generating public/private rsa1 key pair.
Your identification has been saved in /etc/ssh_host_key.
Your public key has been saved in /etc/ssh_host_key.pub.
The key fingerprint is:
af:36:e3:f4:8e:c7:f2:82:1d:23:fb:9f:25:93:97:8d
The key's randomart image is:
+--[RSA1 2048]----+
|                 |
|                 |
|                 |
|                 |
|        S        |
|      . o. . +   |
|       =.+= E .  |
|      o.B+o*     |
|       +=X*      |
+-----------------+
Generating RSA Key...
/usr/bin/ssh-keygen: /usr/lib/libcrypto.so.1.0.0: no version information
available (required by /usr/bin/ssh-keygen)
Generating public/private rsa key pair.
Your identification has been saved in /etc/ssh_host_rsa_key.
Your public key has been saved in /etc/ssh_host_rsa_key.pub.
The key fingerprint is:
11:28:51:a5:5d:33:74:9f:f2:01:c7:62:e9:17:c4:48

～　略　～

|        S .      |
|                 |
|                 |
|                 |
|                 |
+-----------------+
Starting sshd: /usr/sbin/sshd: /usr/lib/libcrypto.so.1.0.0: no version
information available (required by /usr/sbin/sshd)
OK
```

① ネットワーク経由でログインできるようにするために、root のパスワードを設定します。
XG-3358、XG-BBEXT とシリアル接続されたコンソールから「passwd」コマンドを使ってパスワードを設定します。
ここでは便宜上パスワードは「xg3358&bbext」とします。ここで設定したパスワードは Qt のデバイス構成で使用します。

```
# passwd      入力
Changing password for root
New password:       入力
Retype password:    入力
Password for root changed by root
```

② ssh にてログインできることを確認します。
PC（Ubuntu）から ssh コマンドでログインします。最初のログインでは認証のための問い合わせがありますので「yes」と入力します。接続ができればパスワードを聞いてきますのでパスワードを入力します。

```
省略 $ ssh root@192.168.128.200     入力
The authenticity of host '192.168.128.200 (192.168.128.200)' can't be established.
ECDSA key fingerprint is 89:8f:1b:c3:3d:91:e7:59:43:fc:20:00:0a:d0:d4:e5.
Are you sure you want to continue connecting (yes/no)? yes     入力
Warning: Permanently added '192.168.128.200' (ECDSA) to the list of known hosts.
root@192.168.128.200's password:     入力
```

設定が同じ IP の別のターゲットボードボードに接続しようとした場合、PC(Ubuntu)側から見ると偽のボードで不正をしようとしてるのではと警告が表示され接続は拒否されます。

デバッグなどで、同じ IP で行う場合によくあります。
この場合は警告の文章の中に対処するためのコマンドが表示されていますのでその部分をコピー＆ペーストして実行し「~/.ssh/known_host」のホスト情報を削除します。

例： ssh-keygen -f "/home/guest/.ssh/known_hosts" -R 192.168.128.200

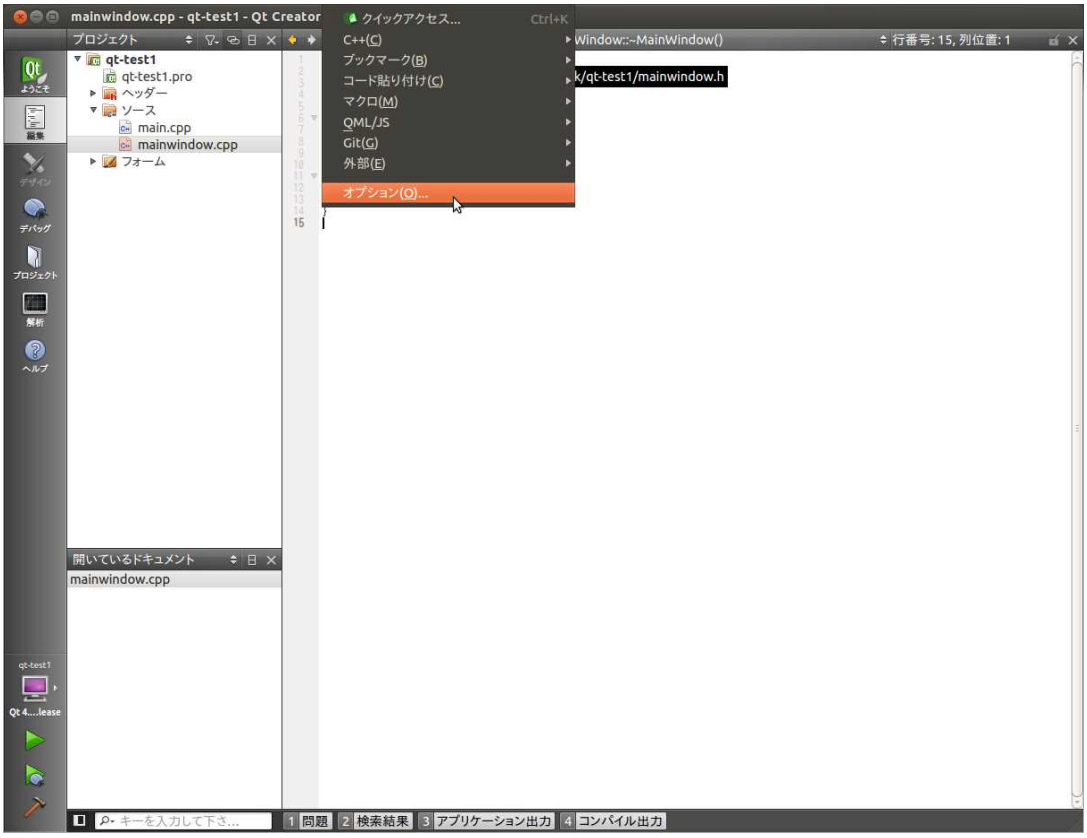
4.3　アプリケーションの作成

GUI アプリケーションを開発するには GUI 用の IDE（統合開発環境）が便利です。ここでは QtCreator を使用してアプリケーションを開発する手順を説明します。QtCreator を使ってのアプリケーション開発については「Qt アプリケーションノート」を参照してください。

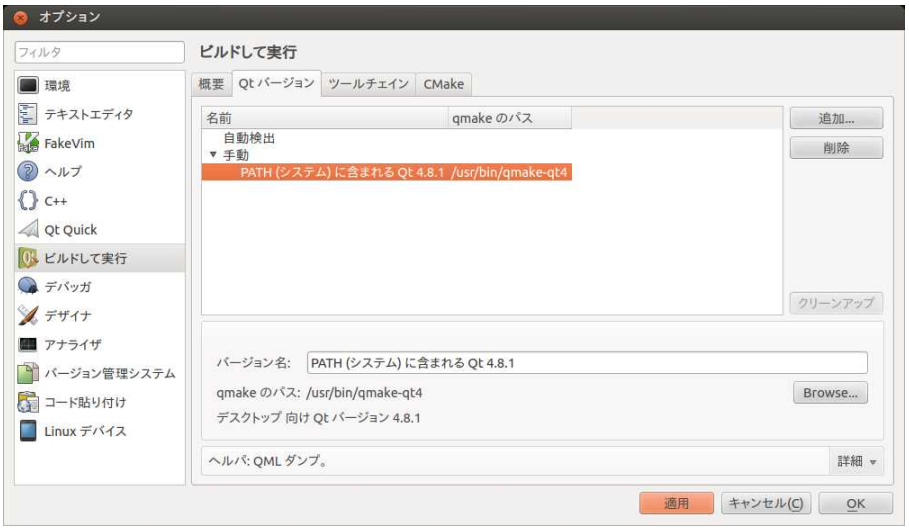
QtCreator XG シリーズのためのデフォルト設定

すでに XG シリーズのためのデフォルト設定されている場合は、次の項目に進んでください。ここではビルドについて最小限の説明しかしてありませんので、配布／デバッグ等詳細の設定はアプリケーションマニュアル「Qt(4.8.5)プログラミング」を参照してください。
XG-3358, XG-BBEXT 用のクロス開発用のデフォルト設定をします。デフォルト設定ですので一度設定するだけです。

①　メニューの「ツール」－「オプション」にてオプションダイアログを開きます。

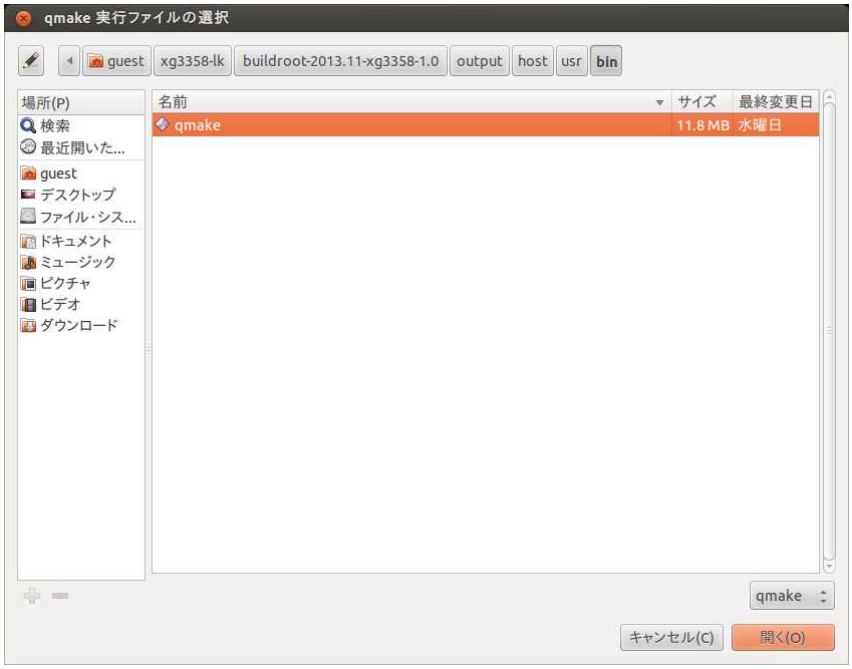


② 「ビルドして実行」を選択し「Qt バージョン」ページを開きます。
XG-3358 または XG-BBEXT の Qt4.8.5 が登録されていなければ「追加ボタン」をクリックします。
登録済みの場合は、手順⑤に進みます。

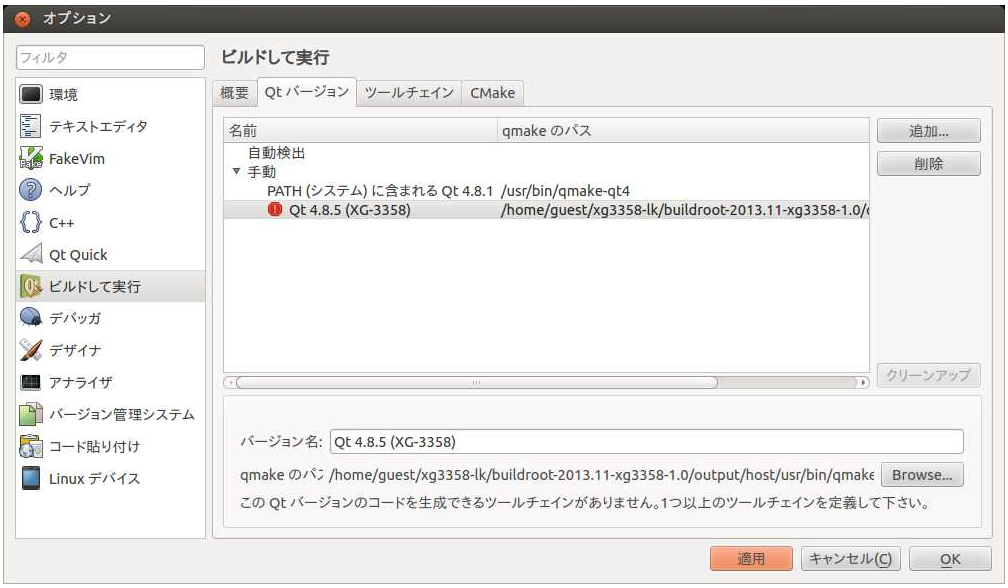


③ 「qmake 実行ファイルの選択」ダイアログにて、XG-3358 または XG-BBEXT の Buildroot システムのホスト qmake を選択します。

機種（開発キット）	qmake のパス
XG-3358(LK-3358-A01)	~/xg3358-lk/buildroot-2013.11-xg3358-X.X/output/host/usr/bin/qmake
XG-BBEXT	~/xgbbext-lk/buildroot-2013.11-xgbbext-X.X/output/host/usr/bin/qmake



④　バージョン名を分かりやすいように「Qt 4.8.5(XG-3358)」または「Qt 4.8.5(XG-BBEXT)」に変更します。「この Qt バージョンのコードを生成できるツールチェインがありません。」と表示されますので、ページを「ツールチェイン」に切替えます。



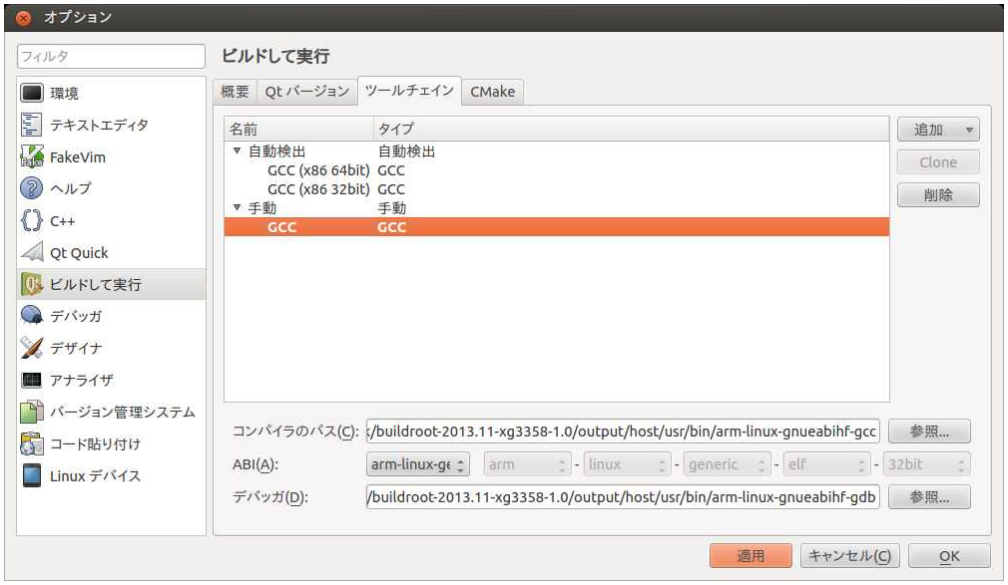
⑤　「ツールチェイン」ページにて追加ボタンをクリックし「gcc」を追加します。手動に追加された GCC 行を選択し、コンパイラのパス、デバッガを「参照」ボタンをクリックして入力します。「適用」ボタンをクリックしツールチェインの登録をし「Qt バージョン」のページに戻ります。

[XG-3358(LK-3358-A01)]

項目	コンパイラのパス
コンパイラ	~/xg3358-lk/buildroot-2013.11-xg3358-X.X/output/host/usr/bin/arm-linux-gnueabihf-gcc
デバッガ	~/xg3358-lk/buildroot-2013.11-xg3358-X.X/output/host/usr/bin/arm-linux-gnueabihf-gdb

[XG-BBEXT]

項目	コンパイラのパス
コンパイラ	~/xgbbext-lk/buildroot-2013.11-xgbbext-X.X/output/host/usr/bin/arm-linux-gnueabihf-gcc
デバッガ	~/xgbbext-lk/buildroot-2013.11-xgbbext-X.X/output/host/usr/bin/arm-linux-gnueabihf-gdb

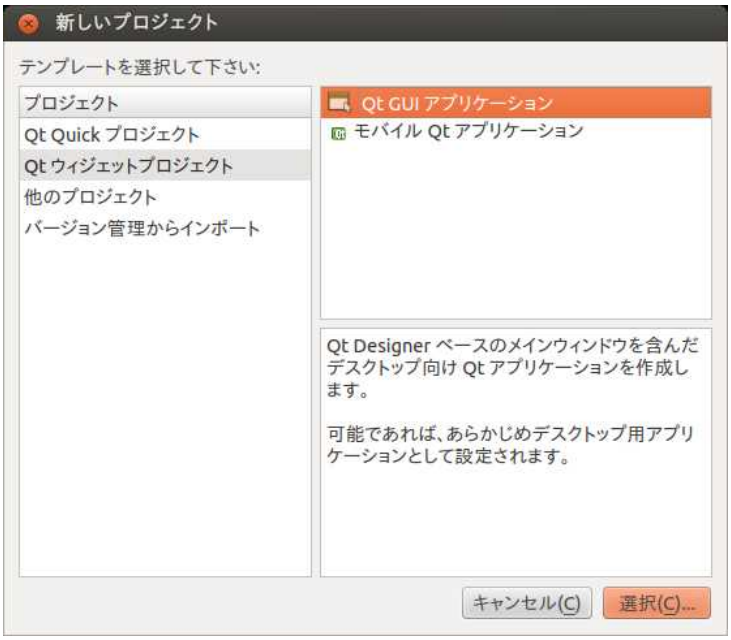


⑥　「Qt バージョン」ページにて先ほどのエラーメッセージが消え「qmlviewer がインストールされていません」というメッセージだけが表示されていることを確認「OK」をクリックします。

cv_sample2 プロジェクトの作成

Qt アプリケーションを作成するには、最初に Qt プロジェクトを以下の手順で作成します。

① メニューの「ファイル」－「ファイル/プロジェクトの新規作成」を選択して「新しいプロジェクト」ダイアログを開きます。
プロジェクトの中の「Qt ウィジェットプロジェクト」「Qt GUI アプリケーション」を選択し「選択(C)…」ボタンをクリックします。



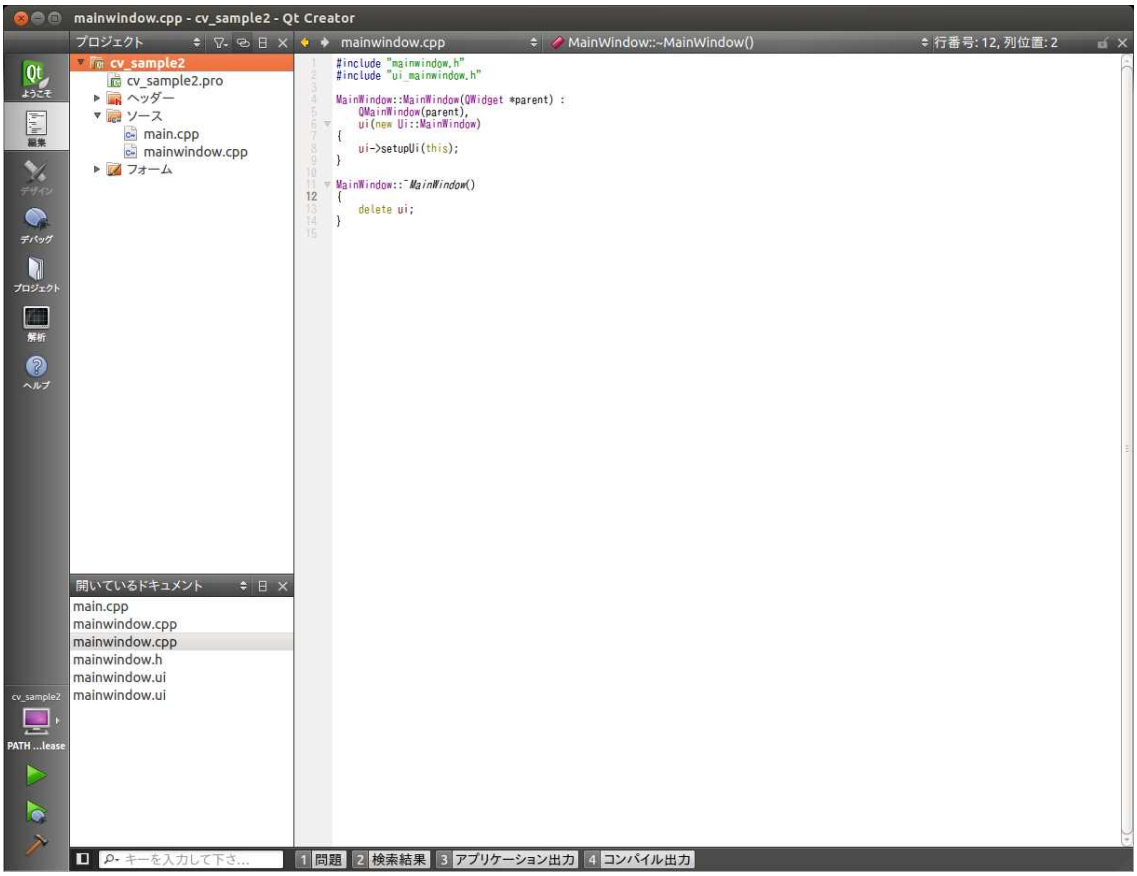
② 名前に「cv_sample2」、パスは参照ボタンをクリックしてディレクトリ「/home/guest/qt-work」を作成または選択し「次へ」ボタンをクリックします。プロジェクトは「/home/guest/qt-work/cv_sample2」に作成されます。

③　ターゲットの設定をします。　デスクトップのチェックを確認し、ビルド構成の作成は「Qt バージョンごとに Debug と Release を 1 つずつ」を選択します。

④　クラス情報にて、クラス名、ヘッダーファイル、ソースファイル、フォームファイルを設定します。ここではデフォルトのまま「次へ」ボタンをクリックします。

⑤　プロジェクトに作成されるファイル一覧が表示されます。確認の上「完了」ボタンをクリックし、プロジェクトの作成を完了します。
バージョン管理システムを使用する場合は「バージョン管理システムに追加」項目でバージョン管理システムを選択してください。

⑥　cv_sample2 プロジェクト　開発用に変わります。



cv_sample2 プロジェクトの設定

Qt アプリケーションプロジェクトの設定をします。

① プロジェクトをクリックし「ビルド設定」ページを選択します。

② 最初に Qt バージョンは「Qt 4.8.5(XG-3358)」を選択します。Qt バージョンの選択を終えると「ビルド構成を編集」は「Qt 4.8.5(XG-3358) Debug」または「Qt 4.8.5(XG-3358) Relase」になりますので「Qt 4.8.5(XG-3358) Debug」を選択します。

シャドウビルドが選択されていると、構成によってビルドされるディレクトリが別々になります。

③ ビルド後、出来上がった実行可能ファイルを/nfs ディレクトリに自動的にコピーするように設定します。
「ビルドステップを追加」ボタンをクリックすると、メニューが表れますので「独自プロセスステップ」を選択します。

④　独自プロセスの「独自プロセスステップの有効化」にチェックを入れ、コマンドに「cp」、コマンド引数に「./cv_sample2 /nfs」と入力します。

⑤　編集ボタンをクリックして、編集モードに戻します。

OpenCV アプリ開発の基本設定

ソースコードを入力する前に XG シリーズで GUI ベースの OpenCV アプリケーションを開発する場合に特有なコードについて説明します。

① プロジェクトファイルの編集
リンクする OpenCV ライブラリモジュールの指定や、リモートデバッグする場合のターゲット側に配置するパスを指定します。

プロジェクトファイル「cv_sample2.pro」の最後に以下のコードを追加します。

```
LIBS += ¥
-lopencv_core ¥
-lopencv_imgproc ¥
-lopencv_highgui

target.path += /root/opencv
INSTALLS += target
```

LIBS はリンクする OpenCV ライブラリモジュールです。ここでは基本的な 3 つのモジュールのみ指定しています。利用する OpenCV ライブラリに応じて、対応するモジュールを指定してください。
target.path はデバッグ機能でビルドした実行可能ファイルを配布する XG ターゲットボード側のパスです。詳細はアプリケーションマニュアル「Qt(4.8.5)プログラミング」を参照してください。

② OpenCV 用のヘッダーファイルの追加
コンパイルするときに必要なヘッダーファイルを追加します。
mainwindows.h に「<opencv2/opencv.hpp>」を追加すれば、OpenCV 関連の必要なファイルはインクルードされます。

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <opencv2/opencv.hpp>

        :
   以下省略
```

③　フレーム無しで LCD に表示するためのコード追加

LCD 表示の場合、表示する部分を囲むフレーム（最大化、最小化ボタンを含む）は必要ありませんので表示しないようにコンストラクタでフレーム無しに設定します。

mainwindow.cpp の「MainWindow::MainWindow(QWidget *parent)」コンストラクタに以下のコードを追加します。

```
    this->setWindowFlags(Qt::FramelessWindowHint);
```

④　namespace の追加

OpenCV 関数を呼び出すごとに「cv.」をつけるのは大変なので namespace で指定します。

```
using namespace cv;
```

⑤　日本語表示のための設定

GUI 設計にて設定した Label や Button の日本語で入力したテキスト文字はそのまま表示されますが、コードで設定した日本語は文字化けしてしまいます。そのため以下のコードをコンストラクタに追加します。

```
QTextCodec::setCodecForTr(QTextCodec::codecForLocale());
```

⑥　XG シリーズと Ubuntu デスクトップにてアプリケーションソースを共有するには。

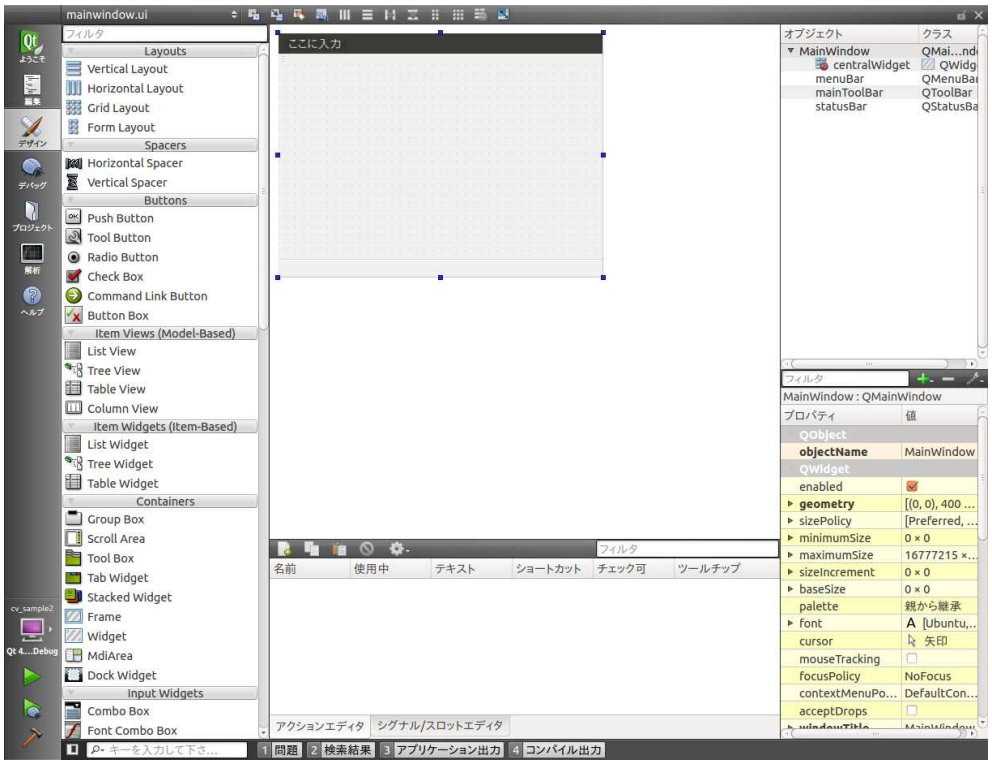
ARM および XG シリーズ特有のコードのあるソースを Ubuntu 側の Qt でビルドするにはその部分を無効にする必要があります。「#ifdef　__arm__ ～　#endif」を使用することにより ARM のみのコードを記述することが可能となります。

②、⑤、⑥を追加したコンストラクタは以下のようになります。

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    QTextCodec::setCodecForTr(QTextCodec::codecForLocale());

#ifdef __arm__
    // ウィンドウのボーダーは何もない Frameless ウィンドウにする。
    this->setWindowFlags(Qt::FramelessWindowHint);
#endif
    ui->setupUi(this);

        :
    途中省略
        :
}
```

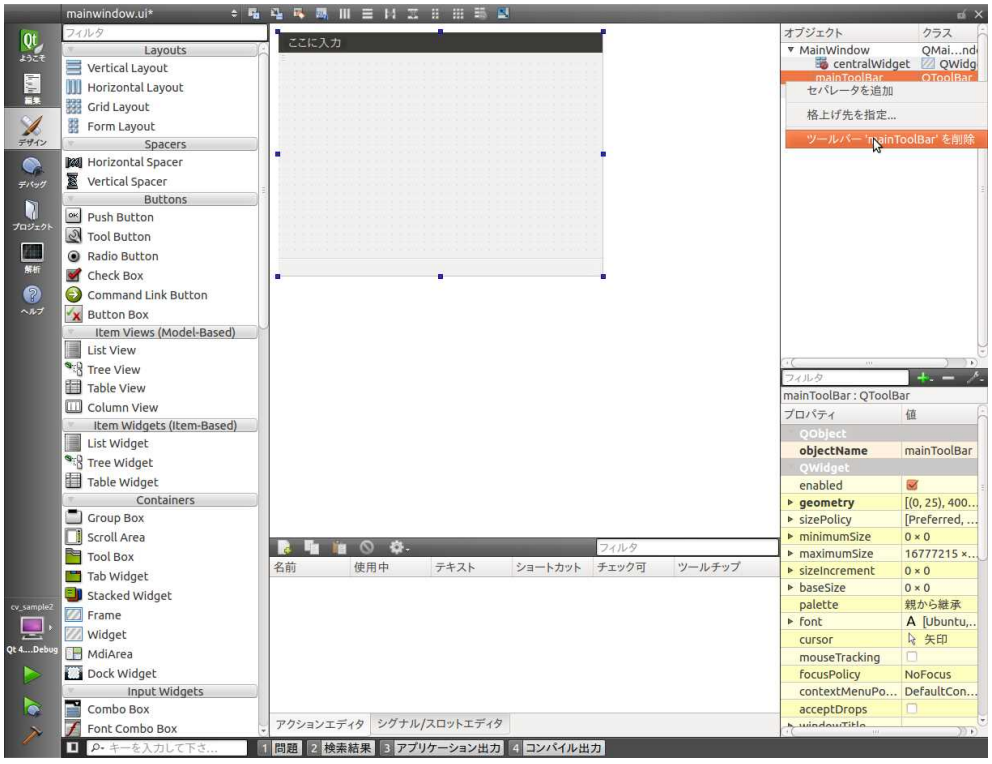
デザイン

① 「cv_sample2」－「フォーム」を展開し「mainwindo.ui」をダブルクリックし GUI デザイン用に切り替えます。

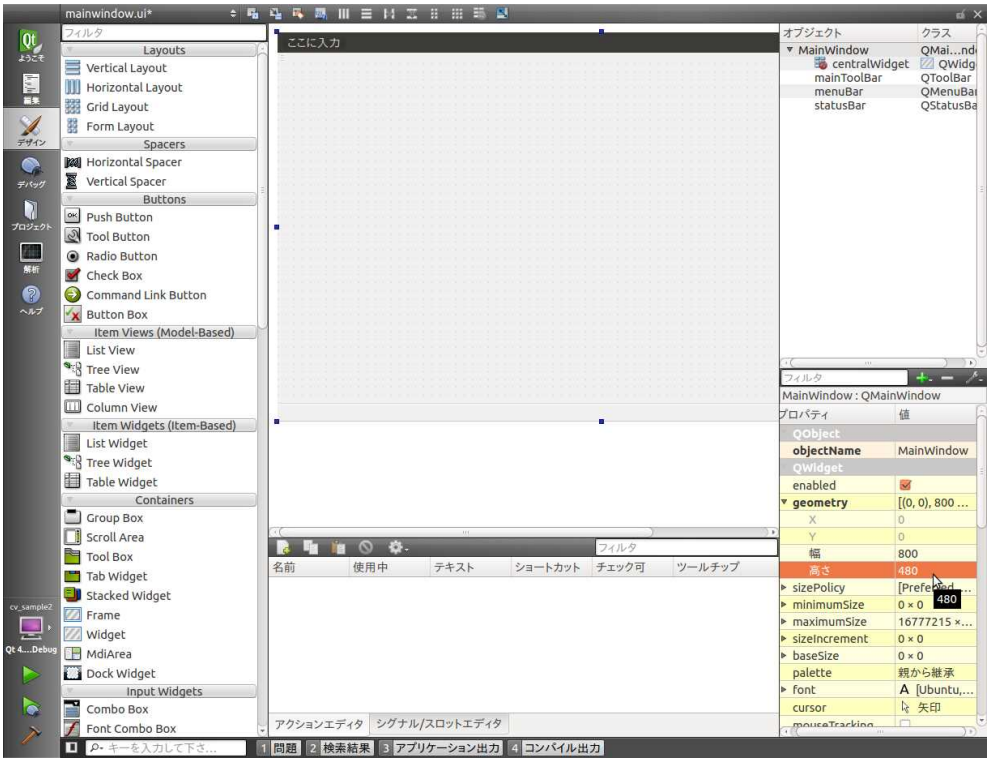


② メニューバー、ツールバー、ステータスバーの削除

LCD 表示では必要としないバーを削除します。オブジェクトインスペクタから「menuBar」「mainToolBar」「statusBar」各々をを選択しマウス右ボタンをクリックしてポップアップメニューの削除を選択します。

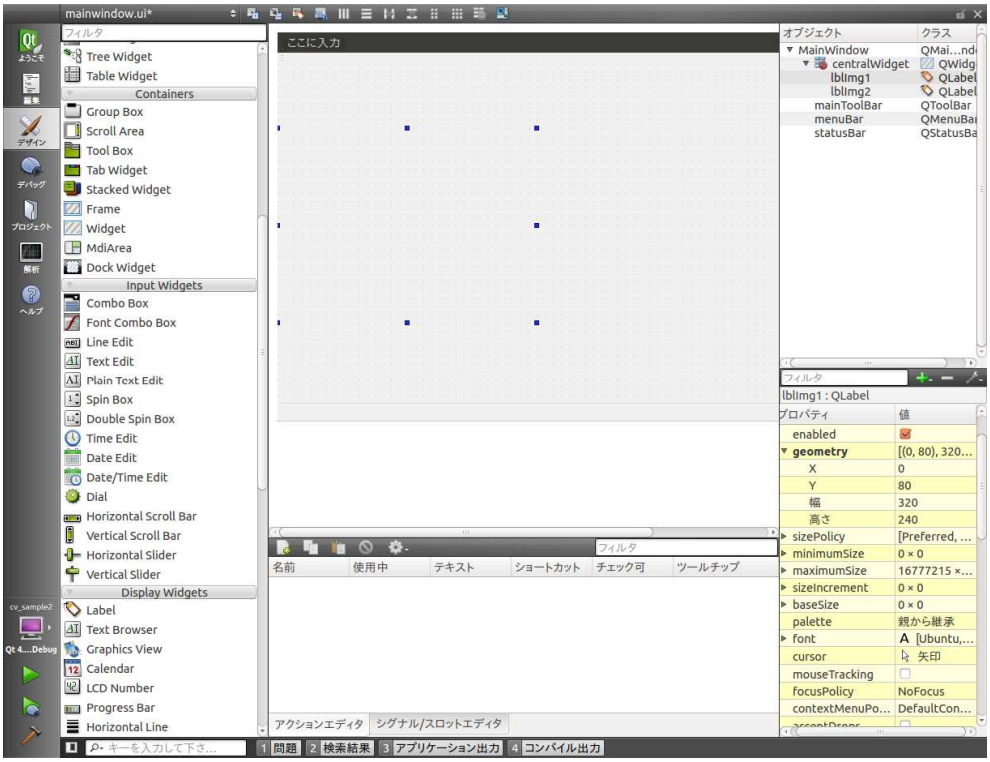


③　Winodow のサイズを指定します。
プロパティの「geometry」で X:0、Y:0、幅：800、高さ:480 に設定します。



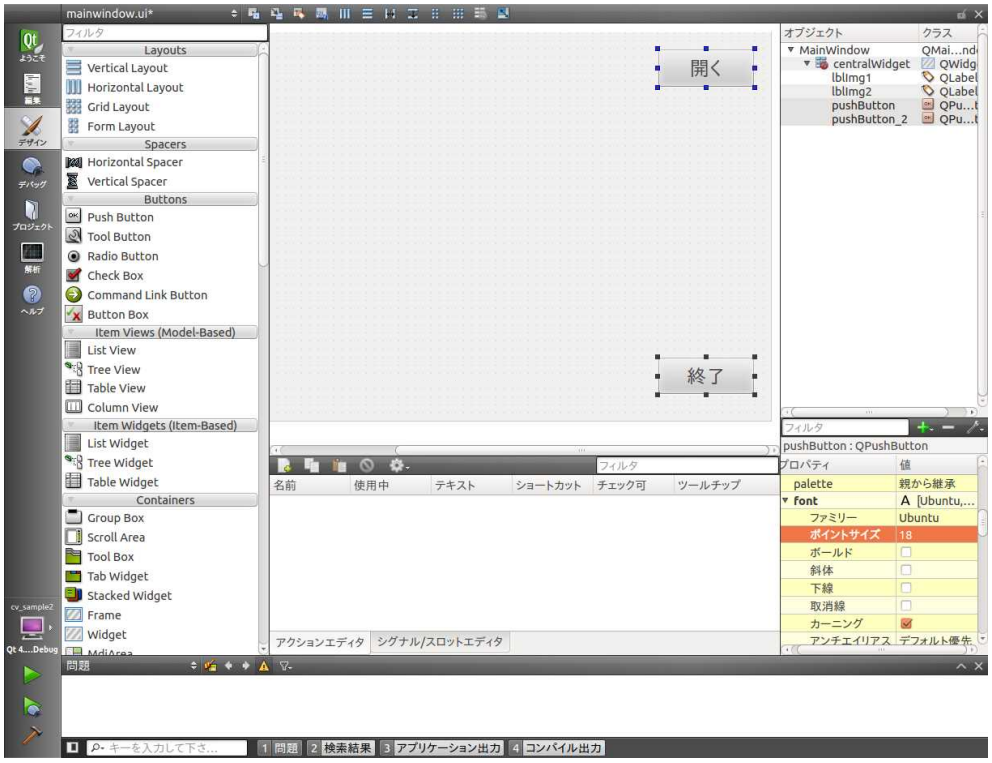
④　Display Widgets から Label を 4 つドラッグ＆ドロップでフォーム左右に配置します。サイズは 320×240 で、左側のラベルは X:0、Y:80　右側のラベルは X:320、Y:80 にプロパティでセットします。

プロパティ	左側 Label	右側 Label
objectName	lblImg1	lblImg2
geometry：X	0	320
geometry：Y	80	80
geometry：幅	320	320
geometry：高さ	240	240
text	（空白）	（空白）
scaledContents	チェック（真）	チェック（真）

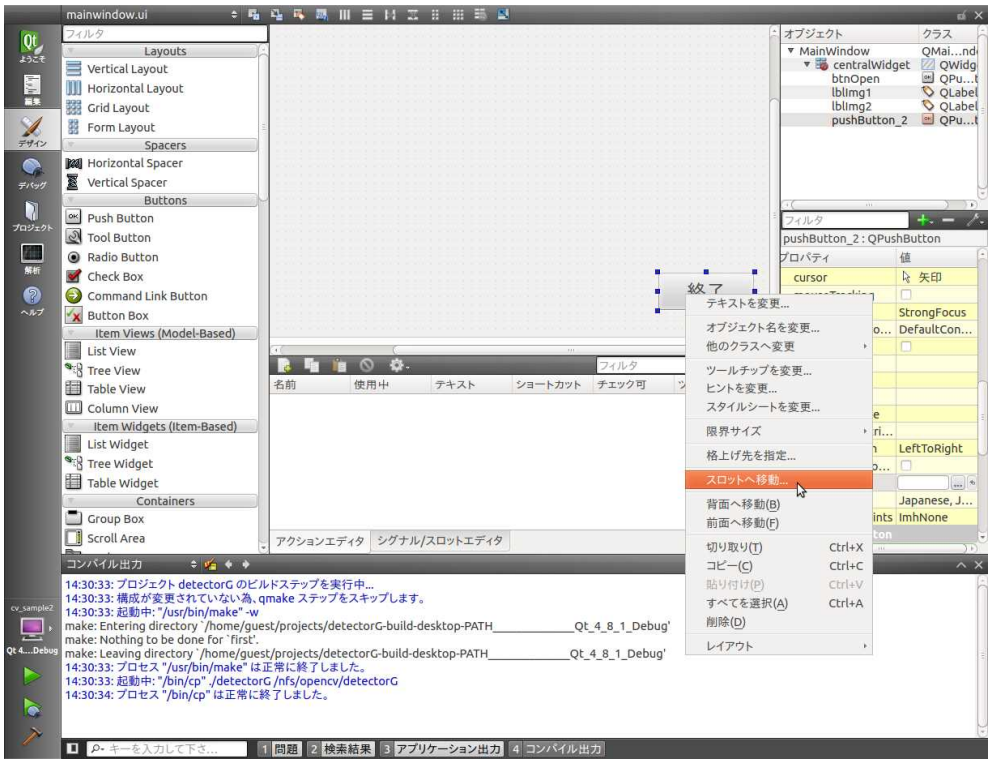


⑤　次にアプリケーションの終了ボタンとファイルを開くボタンを追加します。Buttons の「Push Button」を 2 つフォームの右端の余白にドラッグ＆ドロップで配置します。

プロパティ	終了ボタン	ファイルを開くボタン
objectName	btnQuit	btnOpen
geometry：X	660	660
geometry：Y	20	400
geometry：幅	120	120
geometry：高さ	48	48
text	終了	開く
font：ポイントサイズ	18	18



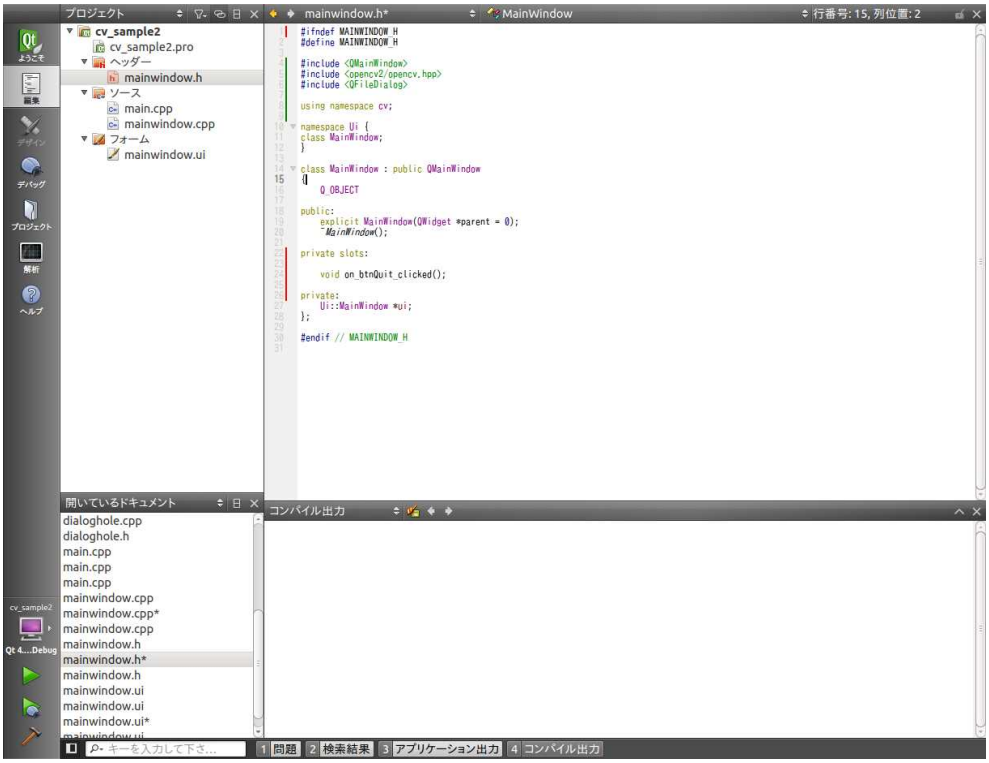
⑥　終了ボタンをタップしたときに、cv_sample2 アプリケーションが終了するようします。そのためには btnQuit にてマウス右ボタンをクリックしポップアップメニューの「スロットへ移動…」を選択します。



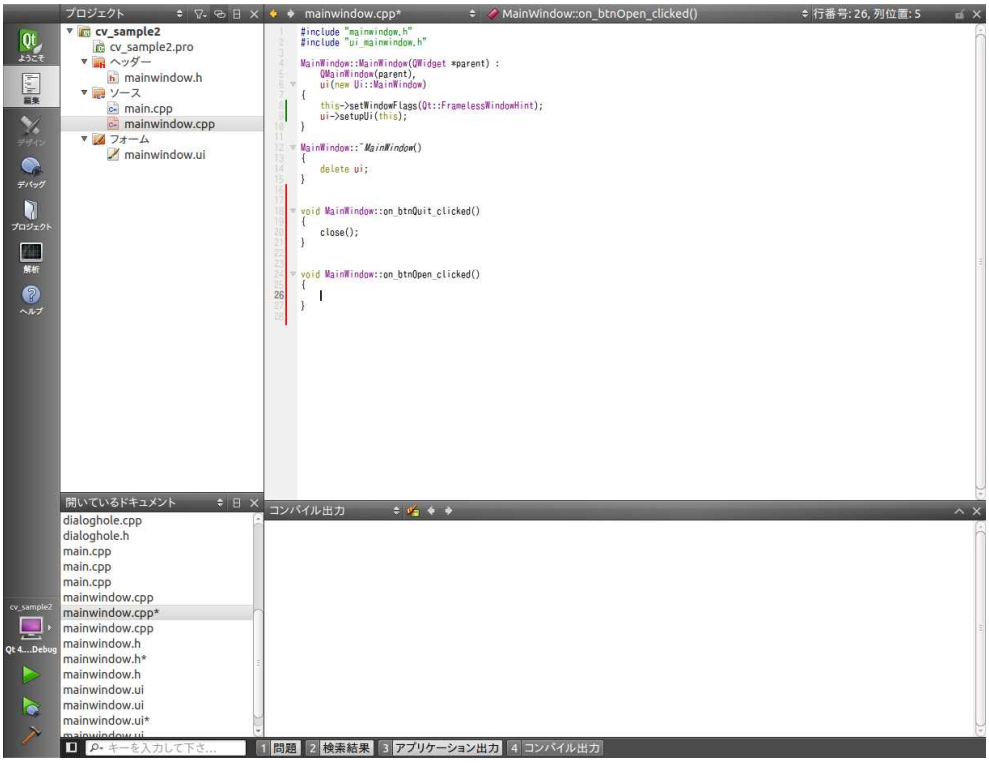
⑦　ここでは単にタップ（クリック）するだけなので、「clicked()」を選択し「OK」ボタンをクリックします。

⑧　自動的にコードエディタに切り替わり「MainWindow::on_btnQuit_clicked()」が自動的に生成されますので、その中に実際の処理のコードを記述します。ここではアプリケーションを終了するために、以下のコードを入力します。

```
close();
```



⑨　ファイルを開くボタン「btnOpen」も、手順⑥～⑦と同様にスロットを作成します。



コードの入力

QtCreator で自動生成されたコードを補完します。

mainwindow.h

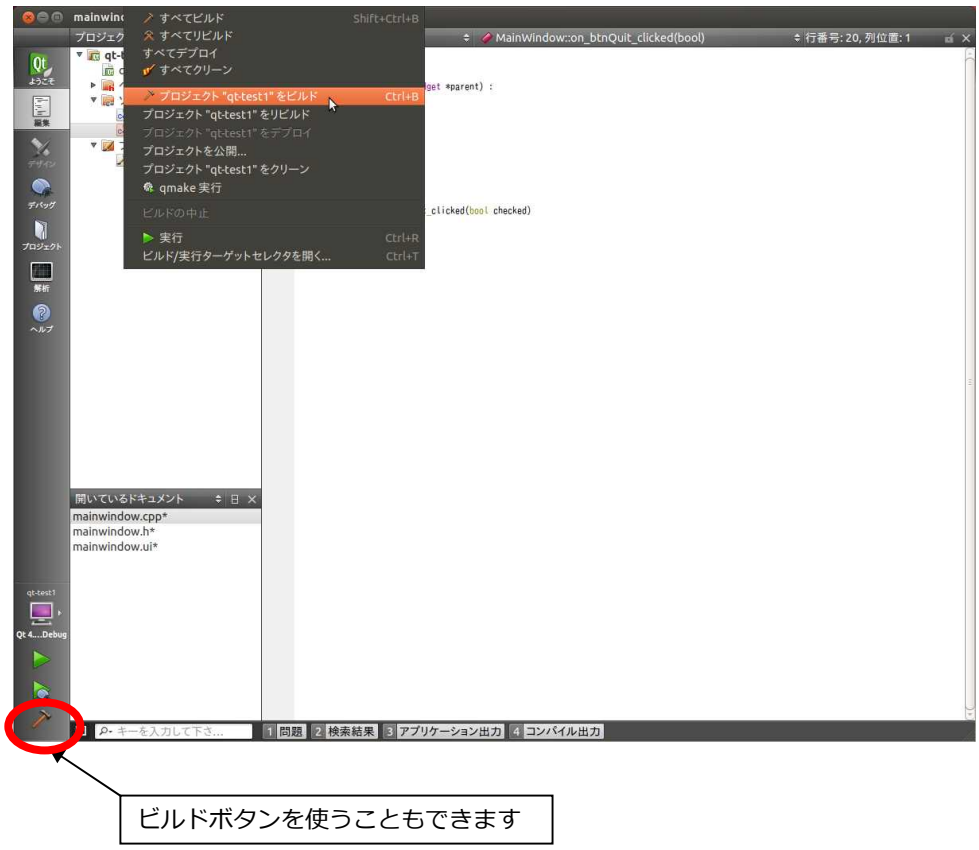
```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <opencv2/opencv.hpp>
#include <QFileDialog>
#include <QTextCodec>
#include <QString>

using namespace cv;

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:

    void on_btnQuit_clicked();
    void on_btnOpen_clicked();

private:
    Ui::MainWindow *ui;
    cv::Mat     img_in, img_in_gray;
    cv::Mat     img_out, img_out_gray;

    QImage      qimg_in, qimg_out;

};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    QTextCodec::setCodecForTr(QTextCodec::codecForLocale());

#ifdef __arm__
    // ウィンドウのボーダーは何もないFramelessウィンドウにする。
    this->setWindowFlags(Qt::FramelessWindowHint);
#endif
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_btnQuit_clicked()
{
    close();
}

void MainWindow::on_btnOpen_clicked()
{
    QString     fileName;

    fileName = QFileDialog::getOpenFileName(this,tr("画像ファイルを開く"),
                        "/root/Images",
                        tr("Image Files [ *.jpg , *.jpeg , *.bmp , *.png , *.gif]"));
    if( ! fileName.isEmpty() ) {
        img_in = cv::imread( fileName.toLocal8Bit().data() );
        if( img_in.data != 0 ) {
            qimg_in = QImage(img_in.data, img_in.cols,
                             img_in.rows,QImage::Format_RGB888).rgbSwapped();
            ui->lblImg1->setPixmap(QPixmap::fromImage(qimg_in));

            cvtColor( img_in, img_in_gray, COLOR_BGR2GRAY );
            Canny( img_in_gray, img_out_gray, 50, 150 );
            cvtColor( img_out_gray, img_out, CV_GRAY2RGB );
            qimg_out = QImage( img_out.data, img_out.cols, img_out.rows,
                               QImage::Format_RGB888).rgbSwapped();
            ui->lblImg2->setPixmap( QPixmap::fromImage(qimg_out) );
        }
    }
}
```

ビルド

Qt アプリケーションのビルドは、Qt Creator メニューの「ビルド」－「プロジェクト"cv_sample2"をビルド」を選択しビルドを開始します。



保存してないファイルがあるときは、以下のように問い合わせがあります。その場合は「すべて保存」をクリックします。「ビルド前にすべてのファイルを保存する」にチェックを入れておけば自動的に保存されてからビルドされるようになります。

ビルドが正常に終了すると、ディレクトリ「/nfs」に「cv_sample2」がコピーされます。

4.4　実行

実行

ビルドが正常に終わると開発Linuxサーバの/nfsディレクトリに実行可能なアプリケーションファイルをコピーされています。
ターゲット側の XG ボードで NFS マウントしてあればそのまま、/mnt/nfs ディレクトリに移動して実行する事ができます。

①　開発用 PC の NFS サーバに接続します。

コマンドで直接 NFS マウントする場合は以下のようにします。

```
# mount -t nfs -o nolock 192.168.128.210:/nfs /mnt/nfs
```
入力

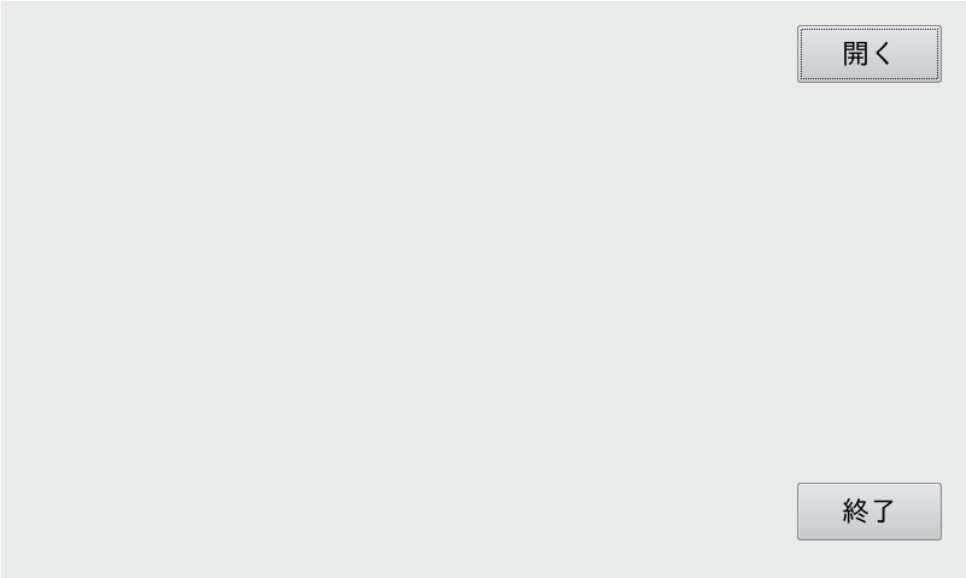
②　qt-test1 が存在することを確認します。

```
# cd /mnt/nfs
# ls cv_sample2
cv_sample2
```
入力

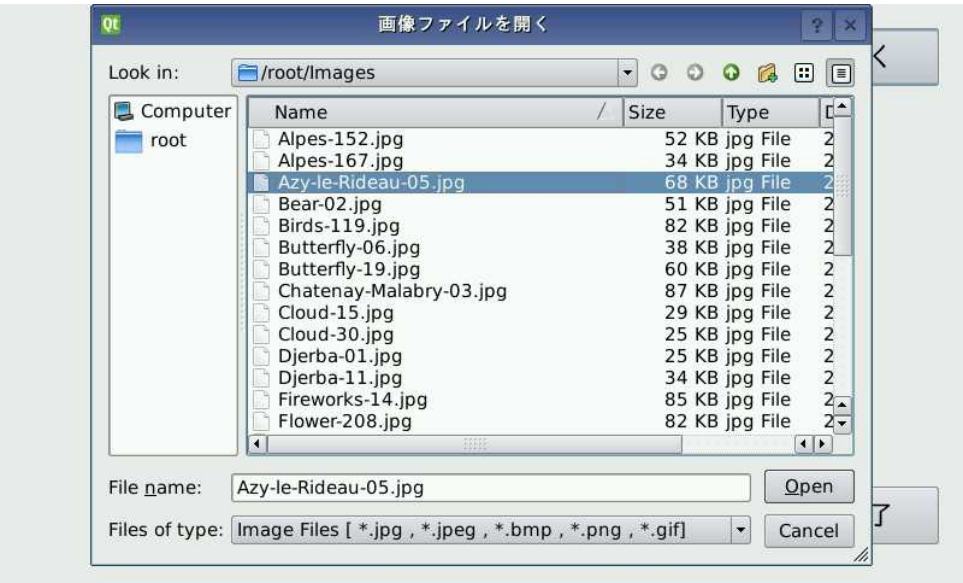
③　qt-test1 を実行します。

```
# ./cv_sample2 -qws
```
入力

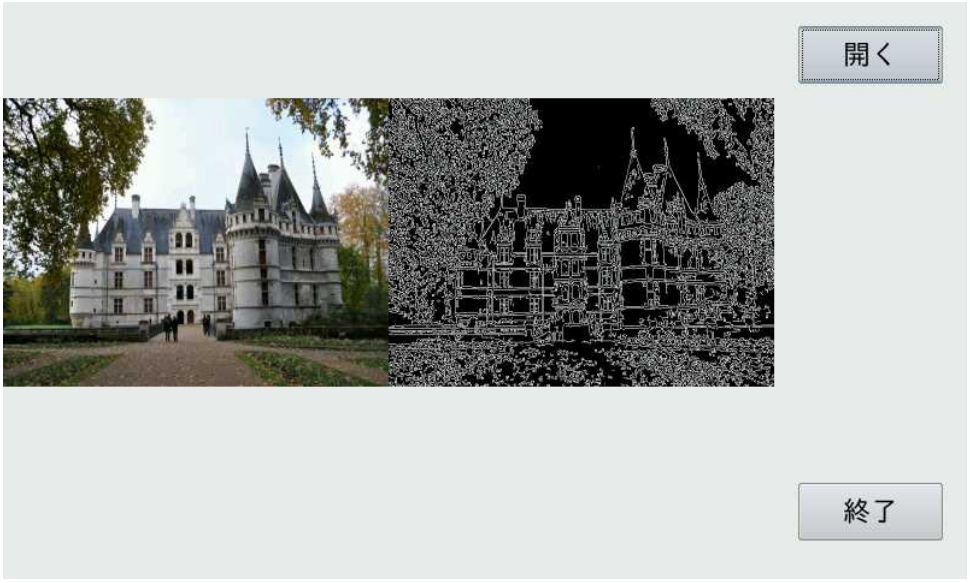
④　次のような画面が表示され「開く」ボタンをタッチするとファイルオープンダイアログが開きます。



⑤　エッジ検出処理を行う画像のファイル選択し「Open」ボタンをタッチします。
テスト用の画像はあらかじめ XG-3358 の /root/Images ディレクトリにコピーしておきます。



⑥　ファイルを読込み、自動的に変換して表示します。



⑦　「終了」ボタンをタッチすると cv_sample2 は終了します。

5.　関連情報

OpenCV に関しての情報は以下の書籍および、サイトが参考になります。

5.1　Web サイト

サイト名	URL
OpenCV	http://opencv.org/
チュートリアル	http://docs.opencv.org/doc/tutorials/tutorials.html
サンプル&ドキュメントの日本語サイト	http://opencv.jp/
OpenCV で学ぶ画像認識	http://gihyo.jp/dev/feature/01/opencv
OpenCV 入門	http://www.buildinsider.net/small/opencv/01
OpenCV 逆引きリファレンス	http://opencv.jp/cookbook/index.html
QtCreator	http://blog.qt.digia.com/jp/2010/03/24/helloworld-from-qt-creator-2/

Table 5.1-1 参考 Web サイト

5.2　書籍

名前	著者	出版社	備考
Learning Opencv: Computer Vision in C++ With the Opencv Library	Gary Bradski Adrian Kaehler	Oreilly & Associates Inc	2014/11/25 発売予定 英文
詳解 OpenCV ―コンピュータビジョンライブラリを使った画像処理・認識	Gary Bradski Adrian Kaehler	オライリージャパン	旧バージョンの OpenCV で解説
実践 OpenCV 2.4―映像処理&解析	永田 雅人	カットシステム	2.4 ではあるが 解説、例は C インターフェース
OpenCV による画像処理入門 (KS 情報科学専門書)	小枝 正直 上田 悦子 中村 恭之	講談社	画像処理入門用

Table 5.2-1 参考書籍

ご注意

商標について

ALPHA PROJECT Co.,LTD.
株式会社アルファプロジェクト
〒431-3114
静岡県浜松市東区積志町 834
http://www.apnet.co.jp
E-MAIL : query@apnet.co.jp