XG Series

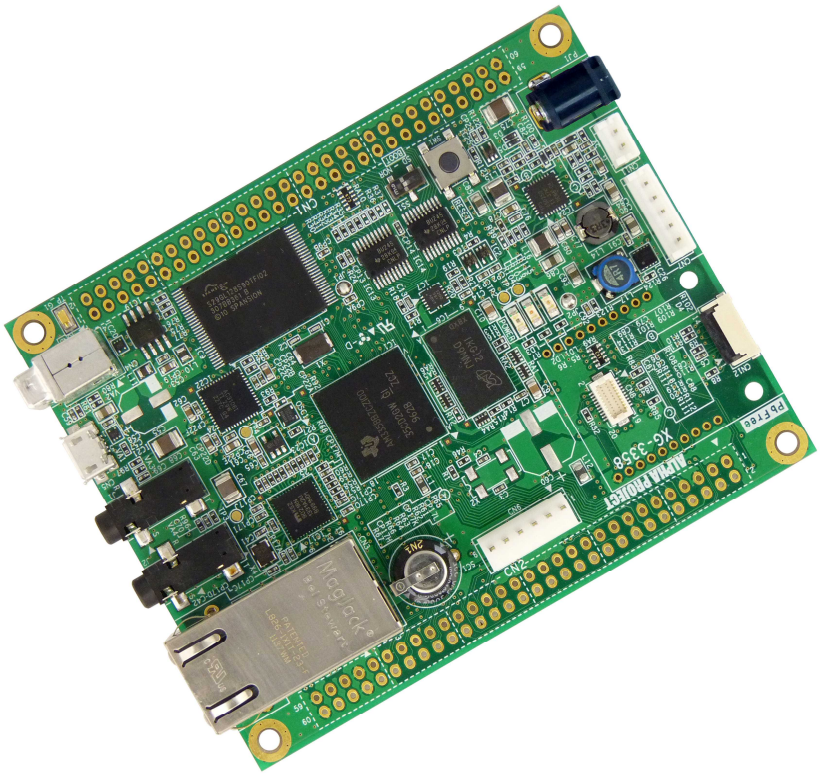
LK-3358-A01

Cortex-A8 AM3358 CPU BOARD

Software Manual

Rev 1.0

ダイジェスト版



ALPHAPROJECT

http://www.apnet.co.jp

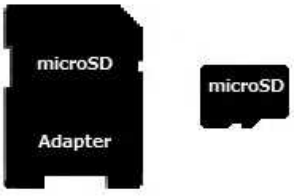
ご使用になる前に

このたびは XG-3358 Linux 開発キット(LK-3358-A01)をお買い上げいただき誠にありがとうございます。
本製品をお役立て頂くために、このマニュアルを十分お読みいただき、正しくお使い下さい。
今後共、弊社製品をご愛顧賜りますよう宜しくお願いいたします。

梱包内容

本製品は、下記の品より構成されております。梱包内容をご確認のうえ、万が一、不足しているものがあれば
お買い上げの販売店までご連絡ください。

LK-3358-A01 梱包内容

●LAN ストレートケーブル　1 本 	●AC アダプタ　1 本 
●PC-USB-04(ケーブル付)　1 個 	●USB ケーブル(A - B)　1 本  コネクタの形状 
●JTAG-CNV-01(ケーブル付)　1 個 	●microSD カード(4GB、アダプター付き)　1 個 
●CD-ROM　　1 枚 ●保証書　　1 枚	

■本製品の内容及び仕様は予告なしに変更されることがありますのでご了承ください。

目　次

1. 概要

1.1 はじめに

XG-3358 は、CPU コアに ARM Cortex-A8 を採用したマイクロプロセッサ「AM3358」(TEXAS INSTRUMENTS)を搭載した汎用 CPU ボードで、標準 OS に Linux を採用しています。
Linux を採用することで、世界中のプログラマによって日々開発される膨大なオープンソースソフトウェア資産をロイヤリティフリーで利用することができます。
本ドキュメントでは、XG-3358 の動作方法をはじめ、SPL、U-Boot、Linux カーネル、アプリケーション開発のための手順を説明します。

本ドキュメントでは、VirtualBox を含めた開発環境が WindowsPC 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。

1.2 Linux について

Linux とは 1991 年に Linus Torvalds 氏によって開発された、オープンソースの UNIX 互換オペレーティングシステムです。
Linux はオープンソース、ロイヤリティフリーという特性から、世界中のプログラマたちにより日々改良され、今では大手企業のサーバーや、行政機関などにも広く採用されています。
また、Linux の特長として CPU アーキテクチャに依存しないということがあげられます。これは、GNU C コンパイラの恩恵にもよるものですが、数多くのターゲット(CPU)に移植されており、デジタル家電製品を中心に非 PC 系製品にも採用されるようになりました。
Linux は、カーネルと呼ばれる OS の核となる部分とコマンドやユーティリティなど多くのソフトウェアから構成されます。これらのソフトウェアの多くは FSF の GNU プロジェクトによるフリーソフトウェアです。
本ドキュメントでは、Linux のごく一部の機能と使い方のみを説明しています。
Linux の詳細については、一般書籍やインターネットから多くの情報を得られますので、それらを参考にしてください。

1.3 U-Boot について

U-Boot は、DENX Software Engineering 社の Wolfgang Denk 氏が保守を行っているオープンソフトウェアの汎用ブートローダです。多くの開発者によって支援され、現在最も機能が豊富で柔軟性に富み、開発が活発に行われています。対応しているアーキテクチャは、SuperH、PPC、ARM、AVR32、MIPS、x86、68k、Nios、MicroBlaze などです。またプログラムのダウンロードに関しても、ネットワークを介した TFTP の他に、CF カード、SD メモリカードなどのストレージデバイスからのダウンロードにも対応しています。

2. システム概要

2.1 システム概要

XG-3358 は、CPU コアに ARM Cortex-A8 を採用したマイクロプロセッサ「AM3358」(TEXAS INSTRUMENTS)を搭載した汎用 CPU ボードです。
Linuxシステムは、ブートローダとLinuxカーネル、ルートファイルシステムから構成されます。ブートローダにSPLとU-Boot、Linux カーネルに Linux-3.2、ルートファイルシステムには RAM,microSD カード等で動作する専用パッケージを使用します。

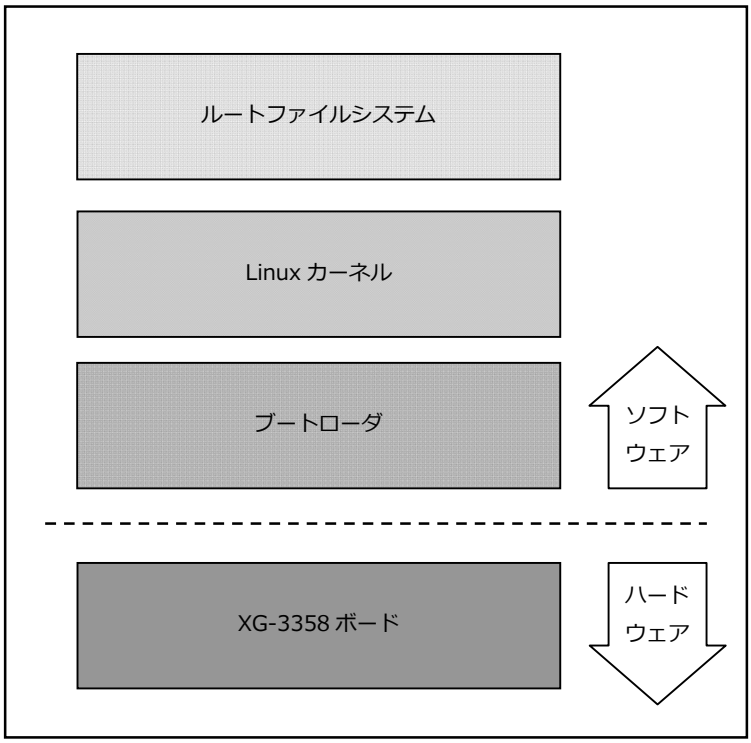


Fig 2.1-1 XG-3358 システム概要図

2.4 ルートファイルシステム

Linux は、カーネルとファイルシステムという 2 つの要素から構成されます。
Linux では、全てのデータがファイルという形で管理されています。アプリケーションプログラムやデバイスドライバをはじめ、HDD や COM ポートなどの入出力デバイスもファイルとして扱われます。
Linux では全てのファイルがルートディレクトリを起点としたディレクトリ構造下に管理されており、これら全てのファイル構造のことをファイルシステムと呼びます。また、システム動作に必要なシステムファイル群のこともファイルシステムと呼びます。
本ドキュメントでは、これらの意味を明確にするため、ファイル管理構造(ext2 や ext3)のことをファイルシステム、システム動作に必要なファイル群のことをルートファイルシステムと表現しています。

Linux のルートファイルシステムは、そのシステムが必要とする機能に合わせて構築する必要があります。
XG-3358 では、以下のルートファイルシステムを用意しています。

●ramfs ルートファイルシステム	RAM 上で動作するように構成されたオリジナル Linux パッケージです。 RAM 上に展開されるため、電源を落とすと変更した内容は破棄されます。
●sd ルートファイルシステム	SD カード用に構成されたオリジナル Linux パッケージです。 ルートファイルシステムが SD カード上に展開されるため、電源を落としても変更した内容は破棄されませんが、電源を落とす前には適切な終了処理が必要になります。
●norflash ルートファイルシステム	NOR FlashROM 用に構成されたオリジナル Linux パッケージです。 ルートファイルシステムが NOR FlashROM 上に展開されるため、電源を落としても変更した内容は破棄されませんが、電源を落とす前に適切な終了処理が必要になります。

本ドキュメントでは、ramfs ルートファイルシステムを利用した Linux システムを RAMFS-Linux システム、sd ルートファイルシステムを利用した Linux システムを SD-Linux システム、norflash ルートファイルシステムを利用した Linux システムを NORFLASH-Linux システムと表現します。

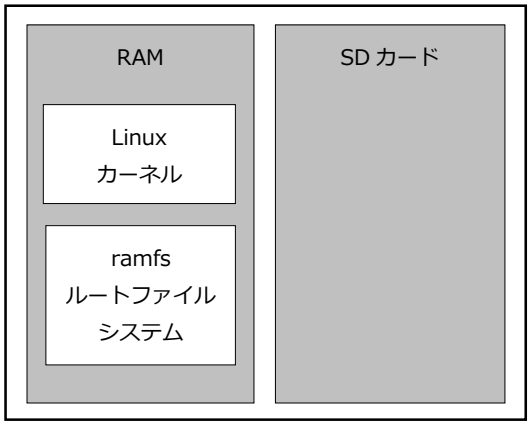


Fig 2.4-1 RAMFS-Linux システム

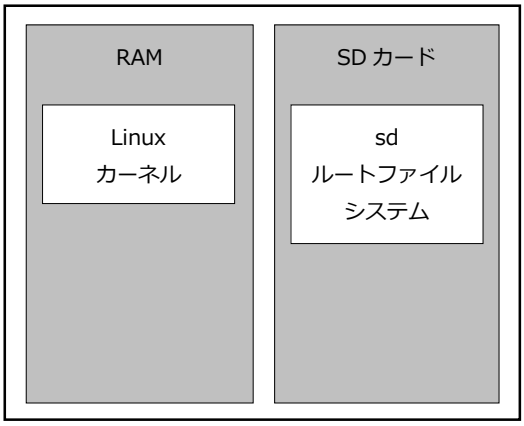


Fig 2.4-2 SD-Linux システム

2.6　添付 CD-ROM の構成

XG-3358 の Linux の開発には、Linux カーネルソース、Buildroot ソースファイル、クロスコンパイラ等が必要です。
これらは、弊社ホームページ及び関連リンクからダウンロードするか、添付 CD-ROM から入手することができます。

```
LK_3358_A01_VX_X
|-- binaries
|   |-- helloworld                          : サンプルアプリ
|   |-- lcdkit                              : タッチパネル LCD サンプルアプリ
|   |-- MLO                                 : SPL バイナリ
|   |-- rootfs.jffs2                        : norflash ファイルシステムバイナリ
|   |-- rootfs.tar.gz                       : sd ファイルシステムバイナリ
|   |-- sample-app                          : サンプルアプリ(デバイス確認用)
|   |-- sample-driver.ko                    : サンプルデバイスドライバ
|   |-- u-boot.bin                          : U-Boot バイナリ(norflash ブート用)
|   |-- u-boot.img                          : U-Boot バイナリ(sd ブート用)
|   |-- uImage-xg3358                       : Linux カーネルバイナリ
|   `-- uInitrd-xg3358                      : ramfs ファイルシステムバイナリ
|-- index.html                              : インデックス HTML
|-- index_images                            : インデックス HTML イメージ
|-- jtag_cnv_01                             : JTAG-CNV-01 関連一式
|-- license
|   |-- fdl.txt                             : GFDL 原文
|   |-- gpl.txt                             : GPL 原文
|   `-- lgpl.txt                            : LGPL 原文
|-- manual
|   |-- lk_3358_a01_sw.pdf                  : LK-3358-A01 ソフトウェアマニュアル
|   `-- lk_install_xg335x.pdf               : Linux 開発 インストールマニュアル
|-- sample
|   |-- devicedriver-X.X.tar.bz2            : サンプルデバイスドライバソース
|   |-- helloworld-X.X.tar.bz2              : サンプルアプリソース
|   `-- lcdkit-X.X.tar.bz2                  : タッチパネル LCD サンプルアプリソース
`-- sources
    |-- buildroot-2013.11-xg3358-X.X.tar.bz2    : Buildroot ソースファイル
    |-- dl-X.X.tar                              : Buildroot ダウンロードファイル一式
    |-- linux-3.2.0-xg3358-X.X.tar.bz2          : Linux カーネルソースファイル
    `-- u-boot-2013.01.01-xg3358-X.X.tar.bz2    : U-Boot ソースファイル
```

Table 2.6-1　CD-ROM 内容

※『X_X』、『X.X』はバージョン番号を示します。バージョン 1.0 の場合は『1_0』、『1.0』になります。

3. システムの動作

3.1 動作環境

Linux の起動を確認するためには、CPU ボードと以下の環境が必要です。

●ホスト PC

Linux では PC をコンソール端末として使用します。

本 Linux 開発キットには、PC-USB-04 が付属しており、PC-USB-04 と PC を USB ケーブルで接続することで、PC 上では仮想シリアルポートとして認識します。

PC-USB-04 の使用方法に関しては、PC-USB-04 のマニュアルをご参照ください。

なお、仮想シリアルポートを使用した通信には、ハイパーターミナル等のターミナルソフトウェアが別途必要となります。

使用機器等	環　　境
CPU ボード	XG-3358
HOST PC	PC/AT 互換機
OS	WindowsVista/7/8
メモリ	使用 OS による
ソフトウェア	ターミナルソフト
USB ポート	1 ポート
LAN ポート	10/100BASE-TX　1 ポート
SD カードスロット	microSD カードを読み込めるスロット(Ubuntu から認識できること)
PC-USB-04	ホスト PC と XG-3358 のシリアル接続用に使用
USB ケーブル	PC-USB-04 で使用
LAN ケーブル	ホスト PC と接続時はクロスケーブルを使用 ハブと接続時はストレートケーブルを使用
Audio 入出力機器	Audio 入出力の動作確認時に使用
LCD-KIT-B01 もしくは LCD-KIT-C01	タッチパネル LCD キットを用いた動作確認時に使用
WM-RP-04S もしくは WM-RP-05S	無線 LAN モジュールを用いた動作確認時に使用
microSD カード	SD ルートファイルシステム作成、MMC ブート確認等に使用
USB ホスト変換ケーブル	XG-3358 の microAB コネクタ(CN5)の動作確認時に使用
USB ファンクション変換ケーブル	XG-3358 の microAB コネクタ(CN5)の動作確認時に使用
PC-CAN-02	CAN 通信の動作確認時に使用
電源	AC アダプタ（DC5V±5%）

Table 3.1-1　動作環境

上記の環境は、XG-3358 の Linux の動作確認をするための環境となります。
カーネル等のコンパイルに使用する開発環境に関しては、開発キット付属の『**Linux 開発 インストールマニュアル**』でご確認ください。

3.5　XG-3358 ボードの接続

ホスト PC と XG-3358 ボードの接続例を示します。
LAN をネットワークと接続する場合は、ネットワーク管理者と相談し、設定に注意して接続してください。

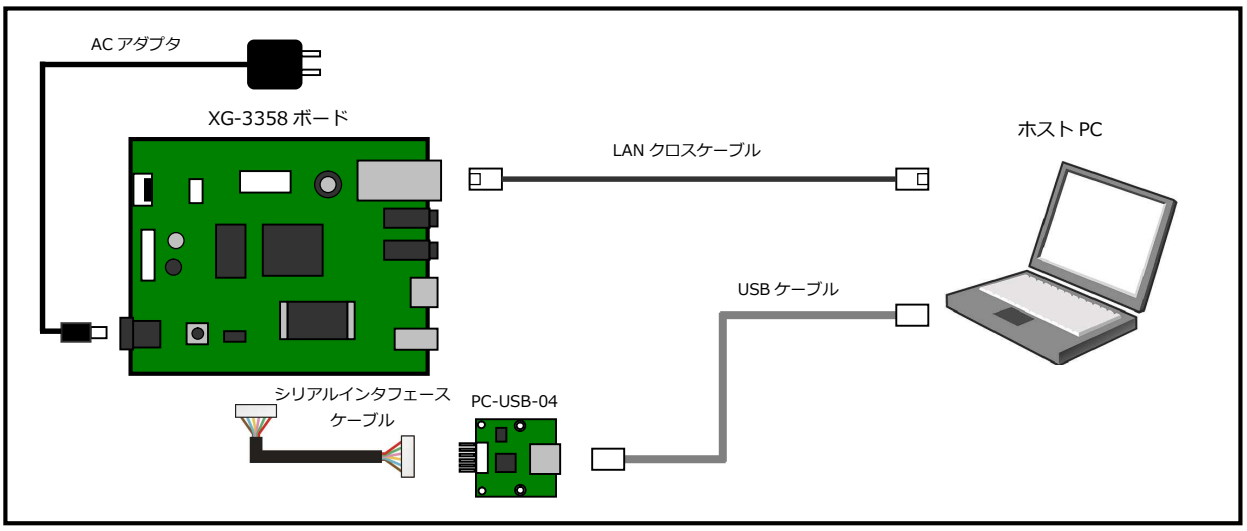


Fig 3.5-1 XG-3358 ボードの接続（PC に接続する場合）

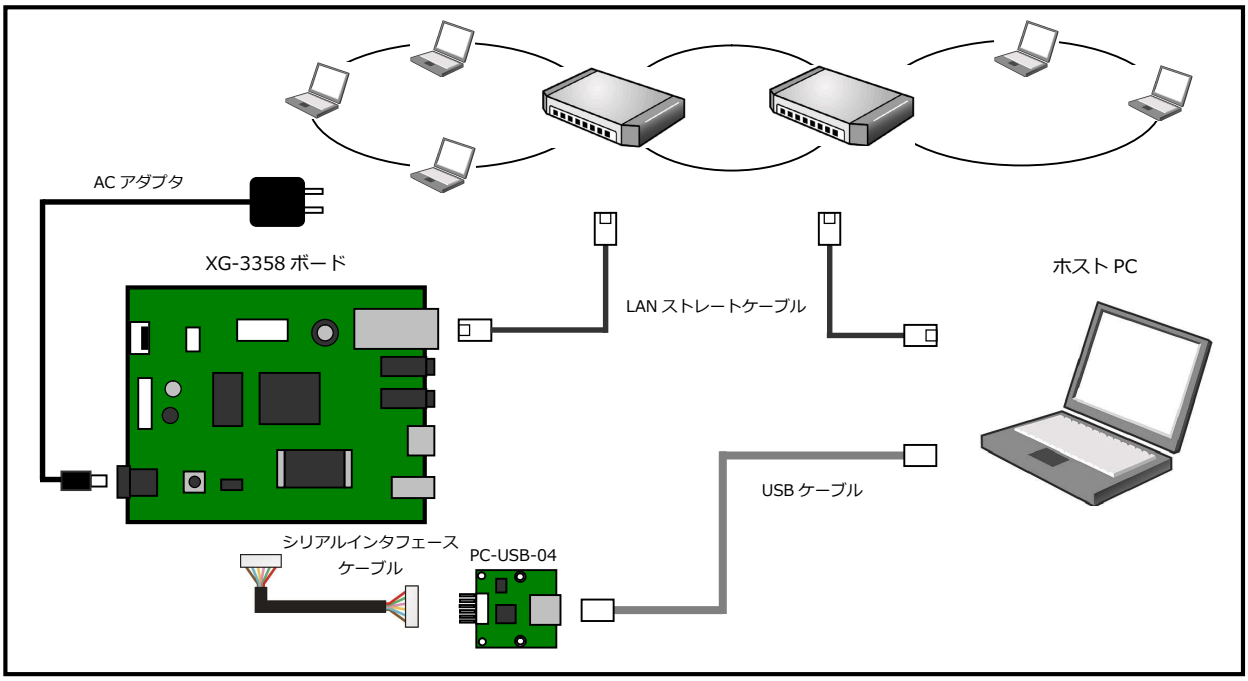
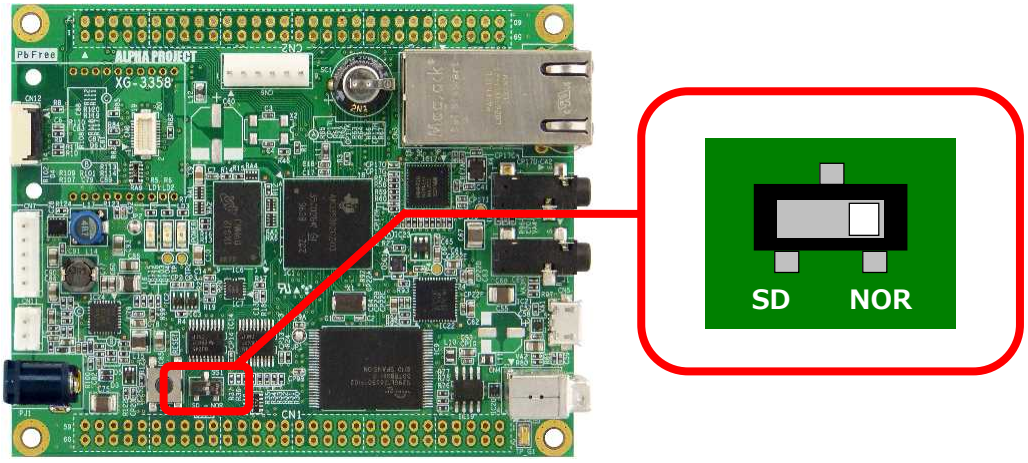


Fig 3.5-2 XG-3358 ボードの接続（HUB に接続する場合）

3.6 Linux の起動

XG-3358 上で Linux の起動を行います。
XG-3358 は、動作確認用に Linux がプリインストールされた状態で出荷しております。

① XG-3358 の電源をいれる前にスイッチが以下の設定になっていることを確認します。
スイッチの設定の詳細に関しては、『**XG-3358 ハードウェアマニュアル**』でご確認ください。



② 『**3.5 XG-3358 ボードの接続**』にしたがって、ホスト PC と XG-3358 を接続します。
PC-USB-04 がホスト PC に認識されて仮想 COM ポートが作成されます。

③ ホスト OS（Windows）のターミナルソフトを起動します。（設定は『**3.2 シリアル初期設定値**』を参照してください）

④ AC アダプタを接続して、XG-3358 の電源を入れます。

⑤ Linux カーネルが自動起動し、全ての起動までにはおよそ 20 秒ほどかかります。
なお、起動ログに関しては、本ドキュメントの『**付録 A. 起動ログ**』でご確認ください。

```
U-Boot 2013.01.01 (Apr 21 2014 - 10:53:03) ALPHAPROJECT XG-3358 vX.X

I2C:   ready
DRAM:  128 MiB

       :
    途中省略
       :

Welcome to Buildroot
xg-3358 login:
```

3.7 Linux の動作確認

XG-3358 上での Linux の動作確認を行います。

ログイン

Linux 起動後、ログインプロンプト『**xg-3358 login:**』が表示されます。
ログインを実行するにはユーザ『**root**』を入力してください。

ログイン設定	
ユーザ	root
パスワード	なし

Table 3.7-1 ログイン設定

```
Welcome to Buildroot
xg-3358 login: root    入力
```

時刻設定

XG-3358 上で時刻の設定をします。XG-3358 には RTC(リアルタイムクロック)が搭載されており、電源を OFF にした状態でも時刻を保持することができます。Linux は起動時に RTC から時刻を読み出し、以後は RTC にアクセスすることなく、CPU 内のタイマーモジュールによって時刻を管理しています。Linux のコマンドライン上から RTC にアクセスするには『**hwclock**』コマンドを使用します。

① RTC に設定されている時刻を読み出すには『**hwclock**』コマンドを引数無しで入力します。

```
# hwclock    入力
Sat Jan  1 12:00:00 2000  0.000000 seconds
```

② RTC に設定されている時刻を変更する際には『**date**』コマンドを使用し、システムの時刻を設定し、その更新されたシステムの時刻を『**hwclock**』コマンドで RTC に書き込みます。
例として時刻を 2014 年 2 月 4 日 15 時 30 分に設定します。
『**date -s '2014-02-04 15:30'**』実行後、『**hwclock -w**』を実行してください。

```
# date -s '2014-02-04 15:30'    入力
Tue Feb  4 15:30:00 UTC 2014
# hwclock -w    入力
```

USB(CN5)　-USB 仮想シリアル-

USB(CN5)を PC に接続することで、PC 上では仮想シリアルとして通信することが可能となります。
なお、PC とは、以下のように接続します。

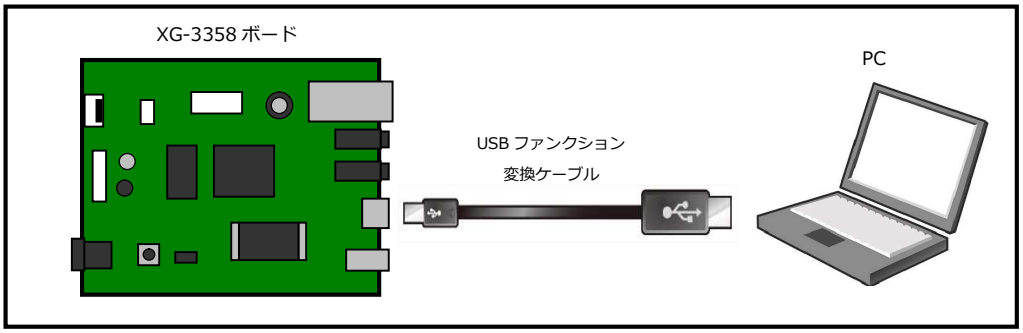


Fig 3.7-3 XG-3358 ボードと PC の接続

PC 側にはデバイスドライバが必要となります。
ドライバのインストール方法に関しては、『**USB 仮想シリアルドライバ インストールガイド**』でご確認ください。

以下に、簡単な動作確認手順を記載します。

① XG-3358 と PC を USB ファンクション変換ケーブルで接続すると PC では仮想シリアルとして認識します。
（仮想シリアルの COM 番号に関しては、『**USB 仮想シリアルドライバ インストールガイド**』でご確認ください。）

② PC で仮想シリアルポートをターミナルソフトで開きます。
通信速度は、参考までに以下に記載しますが、特に指定はないです。

シリアルの設定	
ポート番号	確認した仮想シリアルポート
通信速度	115200bps
データ長	8bit
ストップビット	1bit
パリティ	なし
フロー制御	なし

Table 3.7-2 シリアル設定

③ XG-3358 では、『**/dev/ttyGS0**』を使用しますので、そのポートに対して文字列『**abc**』を送信します。
送信が正常に完了しますと PC のターミナルソフトに表示されます。

```
# echo abc > /dev/ttyGS0    入力
```

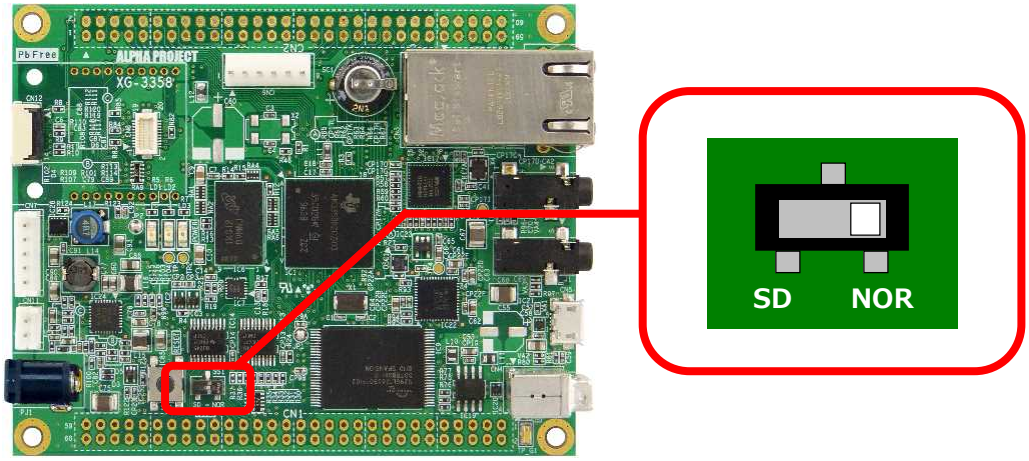
④ XG-3358 の受信では、以下のコマンドで確認できます。
以下のログでは、PC から『**def**』が送信された場合となります。終了する場合には、『**Ctrl+c**』で行います。

```
# cat /dev/ttyGS0    入力
def
```

4.2 ブートローダの起動

XG-3358 を起動して、U-Boot のコマンドコンソールに入る方法を説明します。

① XG-3358 の電源を入れる前に、スイッチが以下のようになっていることを確認します。
スイッチの各設定の詳細に関しては、『**XG-3358 ハードウェアマニュアル**』でご確認ください。



② 『**3.5 XG-3358 ボードの接続**』にしたがって、ホスト PC と XG-3358 を接続します。
PC-USB-04 がホスト PC に認識されて仮想 COM ポートが作成されます。

③ ホスト OS（Windows）のターミナルソフトを起動します。（設定は『**3.2 シリアル初期設定値**』を参照してください）

④ AC アダプタを接続して、XG-3358 の電源を入れます。

4.4 MMC 起動用の U-Boot の作成

ゲスト OS(Ubuntu)上で MMC 起動用の U-Boot をコンパイルするための手順を説明します。

作成の準備

① 作業用ディレクトリ『**xg3358-lk**』をホームディレクトリに作成します。
すでに作成されている場合は、手順②にお進みください。

```
省略 $ mkdir ~/xg3358-lk    入力
```

② 手順①で作成した作業用ディレクトリに移動します。

```
省略 $ cd ~/xg3358-lk    入力
```

③ 作業用ディレクトリに付属 CD 内の以下の 1 つのファイルをコピーします。
手順④～⑥で例として CD から直接コピーする方法を記述します。他の方法でコピーする場合には、コピー作業完了後に、手順⑦にお進みください。

u-boot-2013.01.01-xg3358-X.X.tar.bz2

※『X.X』にはバージョン番号が入ります。Ver1.0 の場合は、『1.0』

④ CD をドライブに挿入します。
デフォルトでは、自動でマウントされますが、マウントされない場合は、以下のコマンドを実行します。

```
省略 $ gvfs-mount -d /dev/sr0    入力
```

マウントされているかどうかは、『**mount**』コマンドで確認できます。
以下のように、『**/dev/sr0**』が表示されている場合は、すでにマウントされています。
（『*******』は、CD のボリュームラベルになります。）

```
省略 $ mount    入力
          :
      途中省略
          :
/dev/sr0 on /media/******* type udf (ro,nosuid,nodev,uhelper=udisks,uid=1000,
gid=1000,iocharset=utf8,umask=0077)
```

⑤ ファイルをコピーします。コマンド途中の『*******』は、CD のボリュームラベルになります。
そのため、その部分は挿入した CD に合わせて入力してください。

```
省略 $ cp /media/*******/sources/u-boot-2013.01.01-xg3358-X.X.tar.bz2 .    入力
```

⑥ CD をアンマウントします。

```
省略 $ umount /dev/sr0    入力
```

⑦ ソースファイルを展開します。

```
省略 $ tar -xjpf u-boot-2013.01.01-xg3358-X.X.tar.bz2    入力
```

作成

① 準備作業で展開した作業用ディレクトリの『**u-boot-2013.01.01-xg3358-X.X**』へ移動します。

```
省略 $ cd ~/xg3358-lk/u-boot-2013.01.01-xg3358-X.X    入力
```

② コンパイルします。

途中の『**O=xg3358**』のＯは、英字大文字のO(オー)ですので、ご注意ください。

```
省略 $ make CROSS_COMPILE=arm-linux-gnueabihf- ARCH=arm O=xg3358 xg3358    入力
Configuring for xg3358 - Board: xg3358, Options: SERIAL1,CONS_INDEX=1
make
make[1]: ディレクトリ `/home/guest/xg3358-lk/u-boot-2013.01.01-xg3358-X.X' に入ります
Generating /home/guest/xg3358-lk/u-boot-2013.01.01-xg3358-X.X/xg3358/include/autoconf.mk

        :
    途中省略
        :

make[2]: `all' に対して行うべき事はありません.
make[2]: ディレクトリ `/home/guest/xg3358-lk/u-boot-2013.01.01-xg3358-X.X/examples/api'
から出ます
make[1]: ディレクトリ `/home/guest/xg3358-lk/u-boot-2013.01.01-xg3358-X.X' から出ます
```

③ make が正常に終了すると『**xg3358**』ディレクトリに『**MLO**』と『**u-boot.img**』が作成されます。

```
省略 $ ls xg3358/MLO xg3358/u-boot.img    入力
xg3358/MLO  xg3358/u-boot.img
```

『**arm-linux-gnueabihf-gcc: コマンドが見つかりません**』のようなエラーが表示される場合には、クロスコンパイラのインストールが正常にできていない可能性があります。
多くの場合でインストールマニュアルの『**SDK インストール手順**』の最後に行うパス設定が異なっている可能性がありますので、再度ご確認ください。

5. Linux

5.1 Linux システムの概要

XG-3358 用 Linux システムは、Linux カーネルとルートファイルシステム(ramfs, sd, norflash)から構成されます。
Linux カーネルは、デバイスドライバとして UART、Ethernet、FlashROM 等をサポートし、ファイルシステムとして ext2、ext3、JFFS2、cramfs、FAT、NFS 等をサポートしています。
ルートファイルシステムは、基本アプリケーションとして、コマンドユーティリティ郡「busybox」が収録されています。

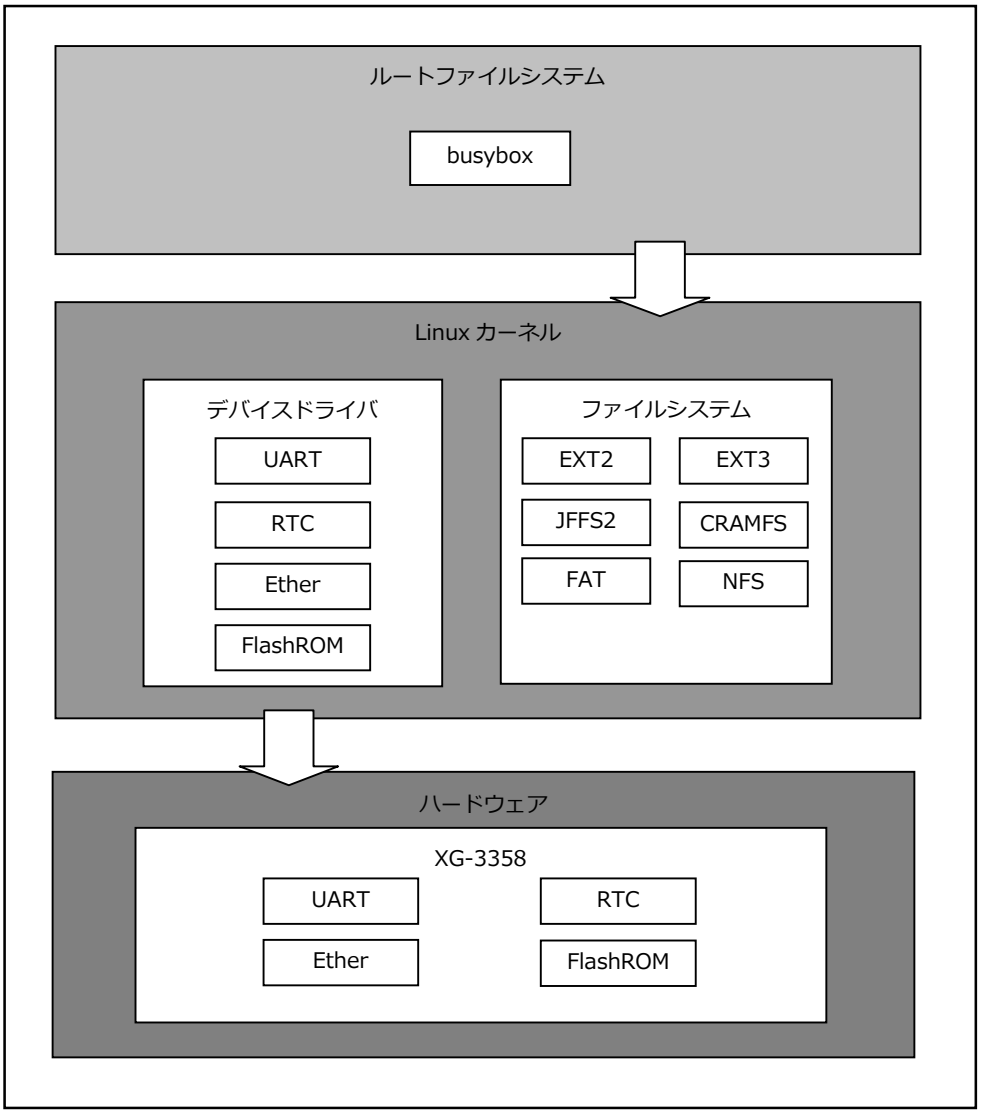


Fig 5.1-1 Linux システム

Linux カーネルの作成

Linux カーネルをコンパイルする方法を説明します。
Linux カーネルの設定データは Linux カーネルソースディレクトリ以下『**arch/arm/configs/xg3358_defconfig**』に保存されています。

① 準備作業で展開した作業用ディレクトリの『**linux-3.2.0-xg3358-X.X**』へ移動します。

```
省略 $ cd ~/xg3358-lk/linux-3.2.0-xg3358-X.X    入力
```

② Linux カーネルの設定データを呼び出します。

```
省略 $ make ARCH=arm xg3358_defconfig    入力
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c

        :
    途中省略
        :

#
# configuration written to .config
#
```

③ make を実行します。終了までに数分から数時間かかる場合があります。

```
省略 $ make ARCH=arm uImage    入力
scripts/kconfig/conf --silentoldconfig Kconfig
  WRAP    arch/arm/include/generated/asm/auxvec.h
  WRAP    arch/arm/include/generated/asm/bitsperlong.h

        :
    途中省略
        :

Load Address: 80008000
Entry Point:  80008000
  Image arch/arm/boot/uImage is ready
```

④ make が正常に終了すると『**./arch/arm/boot**』ディレクトリに Linux カーネルイメージ『**uImage**』が作成されます。

```
省略 $ ls arch/arm/boot/uImage    入力
arch/arm/boot/uImage
```

⑤ 『**/tftpboot**』ディレクトリに Linux カーネルイメージ『**uImage**』をコピーし、名前を『**uImage-xg3358**』に変更します。

```
省略 $ cp arch/arm/boot/uImage /tftpboot/uImage-xg3358    入力
```

『**arm-linux-gnueabihf-gcc: コマンドが見つかりません**』のようなエラーが表示される場合には、クロスコンパイラのインストールが正常にできていない可能性があります。
多くの場合でインストールマニュアルの『**SDK インストール手順**』の最後に行うパス設定が異なっている可能性がありますので、再度ご確認ください。

ramfs ルートファイルシステムの作成

① ルートファイルシステムの make が正常に終了すると、『**./output/images**』ディレクトリに『**rootfs.cpio.gz**』ファイルが作成されます。

```
省略 $ ls output/images/rootfs.cpio.gz      入力
output/images/rootfs.cpio.gz
```

② U-Boot 用のルートファイルシステムに変換します。

```
省略 $ mkimage -A arm -O linux -T ramdisk -C gzip -d output/images/rootfs.cpio.gz outpu
t/images/uInitrd-xg3358    入力
Image Name:
Created:      Fri Apr 18 14:30:05 2014
Image Type:   ARM Linux RAMDisk Image (gzip compressed)
Data Size:    6035319 Bytes = 5893.87 kB = 5.76 MB
Load Address: 00000000
Entry Point:  00000000
```

③ TFTP を使用してロードできるように、『**/tftpboot**』ディレクトリにルートファイルシステム『**uInitrd-xg3358**』をコピーします。

```
省略 $ cp output/images/uInitrd-xg3358 /tftpboot      入力
```

5.4　RAMFS-Linux システムの起動

U-Boot を使用し、『**5.2　Linux カーネルの作成**』で作成した Linux カーネル『**uImage-xg3358**』と ramfs ルートファイルシステム『**uInitrd-xg3358**』をネットワーク経由(TFTP)でダウンロードし RAMFS-Linux システムを起動する方法を示します。

① Linux カーネルイメージ『**uImage-xg3358**』を RAM 上にダウンロードします。

```
U-Boot# tftp 83000000 uImage-xg3358    入力
link up on port 0, speed 1000, full duplex
Using cpsw device
TFTP from server 192.168.128.210; our IP address is 192.168.128.200
Filename 'uImage-xg3358'.

         :
     途中省略
         :

done
Bytes transferred = 2661144 (289b18 hex)
```

② ramfs ルートファイルシステム『**uInitrd-xg3358**』を RAM 上にダウンロードします。

```
U-Boot# tftp 85000000 uInitrd-xg3358    入力
link up on port 0, speed 1000, full duplex
Using cpsw device
TFTP from server 192.168.128.210; our IP address is 192.168.128.200
Filename 'uInitrd-xg3358'.

         :
     途中省略
         :

done
Bytes transferred = 6035383 (5c17b7 hex)
```

③ ダウンロードしたイメージを起動します。

```
U-Boot# bootm 83000000 85000000    入力
## Booting kernel from Legacy Image at 83000000 ...
   Image Name:   Linux-3.2.0-xg3358-X.X
   Image Type:   ARM Linux Kernel Image (uncompressed)

         :
     途中省略
         :

Welcome to Buildroot
xg-3358 login:
```

6. プログラムの作成

本章では、XG-3358 上で動作するアプリケーションの作成方法について説明します。

6.1 プログラムの開発について

ソースファイルのコンパイルから動作までの一連の流れを示します。

① ゲスト OS 上でソースファイルを作成。
② ゲスト OS 上でソースファイルをクロスコンパイルし、実行ファイルを作成。
③ XG-3358 ボード上でゲスト OS を nfs でマウントし、実行ファイルをダウンロード。
④ XG-3358 ボード上で動作を確認。

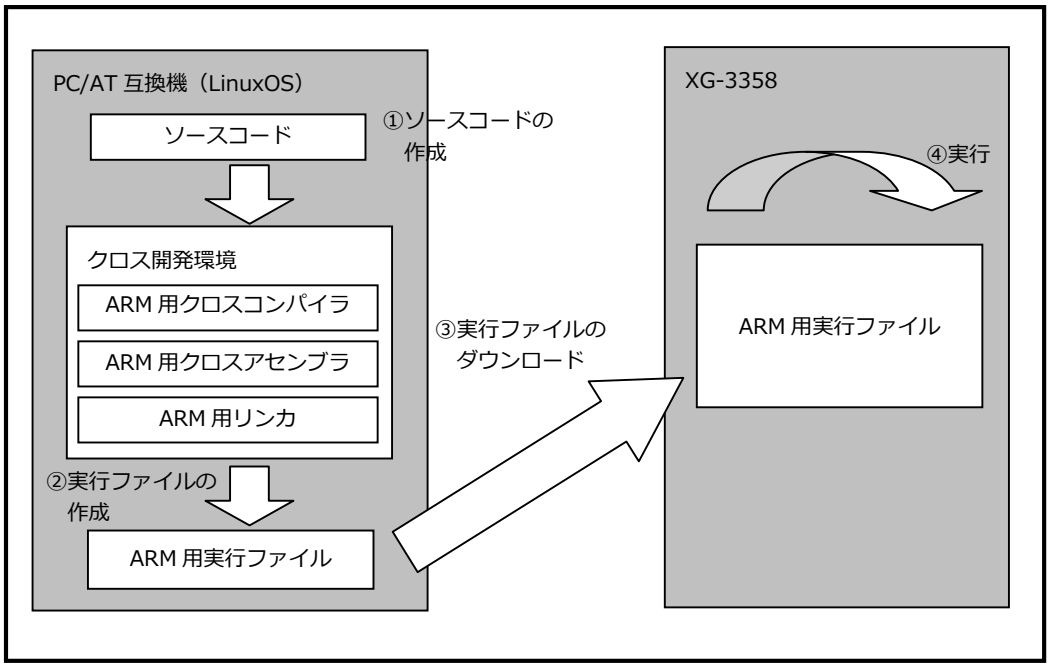


Fig 6.1-1 プログラムの開発手順

サンプルアプリケーションのコンパイル

サンプルアプリケーションのコンパイル手順を説明します。

① 準備作業で展開した作業用ディレクトリの『**helloworld**』へ移動します。

```
省略 $ cd ~/xg3358-lk/helloworld      入力
```

② サンプルアプリケーションをコンパイルします。

```
省略 $ make      入力
arm-linux-gnueabihf-gcc -Wall    helloworld.c   -o helloworld
```

③ アプリケーションプログラムを NFS の共有ディレクトリにコピーします。

```
省略 $ cp helloworld /nfs      入力
```

7. デバイスドライバの作成

本章では、XG-3358 上の LED にアクセス可能なサンプルデバイスドライバの作成方法とそのデバイスドライバを使用したアプリケーションの作成方法について説明します。

本章で作成するプログラムは、Linux カーネルソースが事前にコンパイル済みである必要があります。
カーネルのコンパイルについては、『**5.2 Linux カーネルの作成**』をご確認ください。

7.1 サンプルデバイスドライバの概要

サンプルデバイスドライバは LED デバイスへのアクセス関数を提供します。

デバイスドライバの概要

ユーザプログラム上からデバイスにアクセスする際、通常はデバイスファイルを通じてシステムコールを発行し、デバイスドライバに処理を依頼します。デバイスドライバはデバイスへのアクセス関数を提供することにより、ユーザプログラム上からデバイスにアクセスする手段を提供します。
サンプルデバイスドライバはキャラクタ型デバイスドライバになり、モジュールとしてコンパイルします。このデバイスドライバは、ユーザプログラム上から LED デバイスにアクセスするための関数を提供します。システムコール（API）は『**open**』、『**close**』、『**write**』になります。サンプルデバイスドライバを示すデバイスファイルは『**/dev/sample0**』になります。

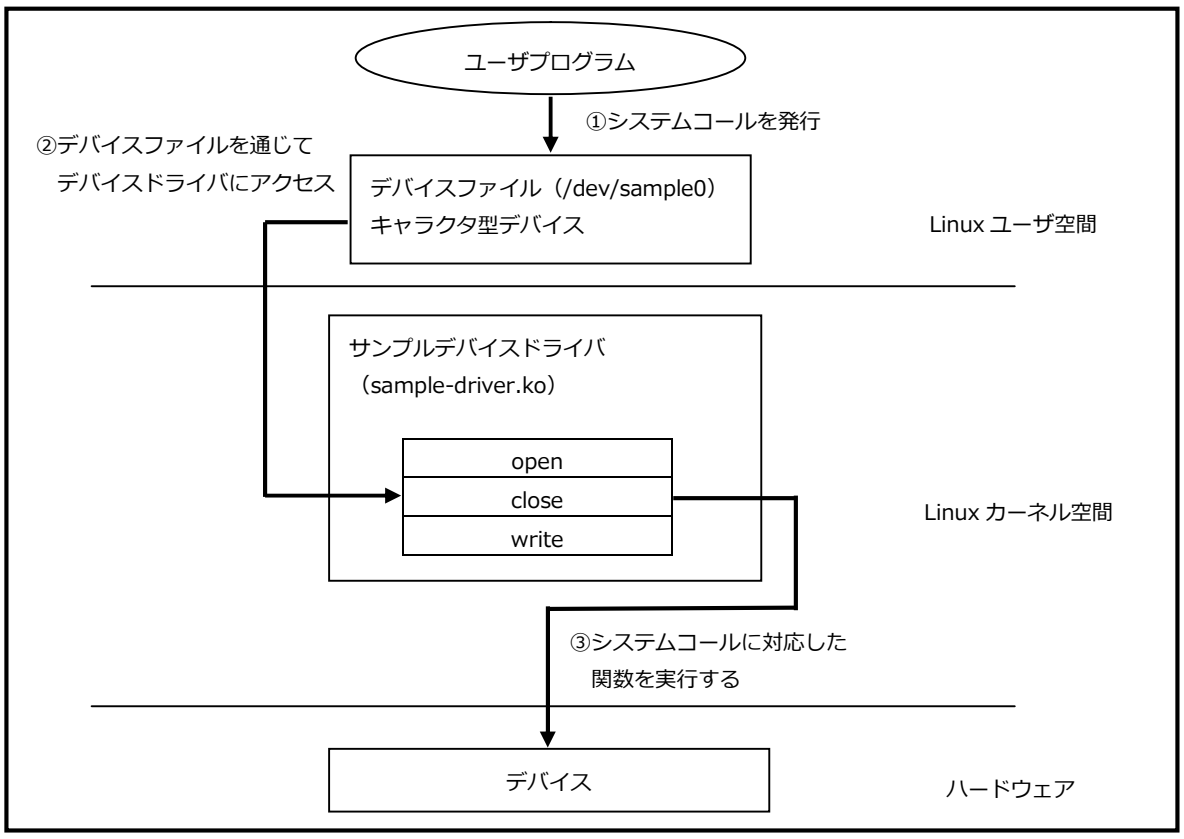


Fig 7.1-1 サンプルデバイスドライバの概要

サンプルデバイスドライバのコンパイル

サンプルデバイスドライバのコンパイル手順を説明します。

① 準備作業で展開した作業用ディレクトリの『**devicedriver/driver**』へ移動します。

```
省略 $ cd ~/xg3358-lk/devicedriver/driver    入力
```

② 汎用デバイスドライバをコンパイルします。

```
省略 $ KBUILD=~/xg3358-lk/linux-3.2.0-xg3358-X.X make    入力
make ARCH=arm -C /home/guest/xg3358-lk/linux-3.2.0-xg3358-X.X M=/home/guest/xg3358-lk/de
vicedriver/driver modules
make[1]: ディレクトリ `/home/guest/xg3358-lk/linux-3.2.0-xg3358-X.X' に入ります
  CC [M]  /home/guest/xg3358-lk/devicedriver/driver/sample-driver.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/guest/xg3358-lk/devicedriver/driver/sample-driver.mod.o
  LD [M]  /home/guest/xg3358-lk/devicedriver/driver/sample-driver.ko
make[1]: ディレクトリ `/home/guest/xg3358-lk/linux-3.2.0-xg3358-X.X' から出ます
```

KBUILD は、Linux カーネルのソースディレクトリを指定します。
カーネルのコンパイルについては、『**5.2 Linux カーネルの作成**』をご確認ください。

③ 作成した汎用デバイスドライバを NFS の共有ディレクトリにコピーします。

```
省略 $ cp sample-driver.ko /nfs    入力
```

サンプルアプリケーションのコンパイル

サンプルアプリケーションのコンパイル手順を説明します。

① 準備作業で展開した作業用ディレクトリの『**devicedriver/application**』へ移動します。

```
省略 $ cd ~/xg3358-lk/devicedriver/application    入力
```

② サンプルアプリケーションをコンパイルします。

```
省略 $ make    入力
arm-linux-gnueabihf-gcc -Wall    sample-app.c    -o sample-app
```

③ アプリケーションプログラムを NFS の共有ディレクトリにコピーします。

```
省略 $ cp sample-app /nfs    入力
```

8. タッチパネル LCD キットの使用

本章では、XG-3358 に LCD-KIT-B01 もしくは LCD-KIT-C01 を接続して動作を行う方法を説明します。

8.1 Linux カーネルの対応方法

Linux カーネルのデフォルトでは、LCD-KIT-B01 もしくは LCD-KIT-C01 を使用する設定になっておりませんので、Linux カーネルを再作成する必要があります。
再作成する手順を以下に説明します。

本手順では、『5.2 Linux カーネルの作成』によって一度 Linux カーネルが作成されていることを前提で説明します。一度も行っていない場合は、一度作成手順を行ってください。

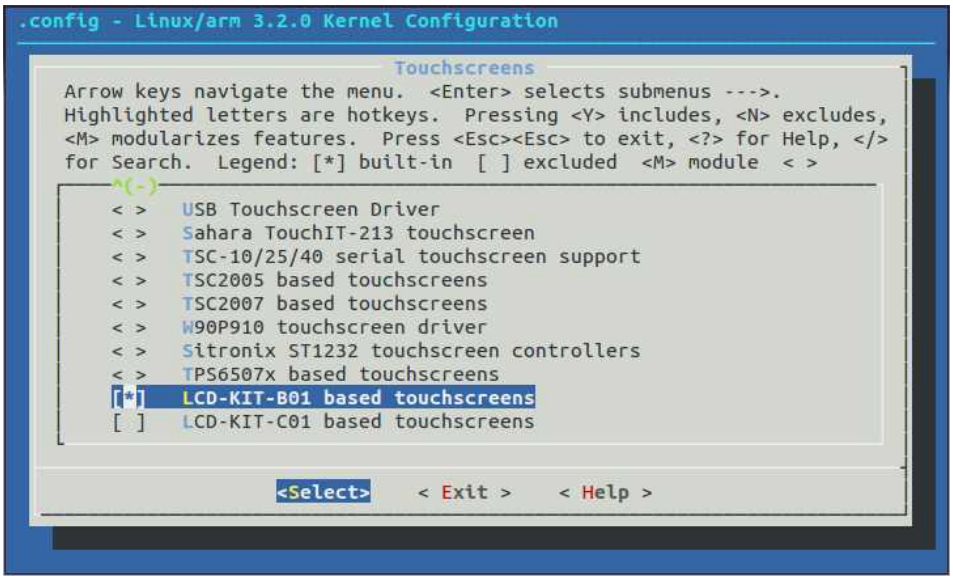
① 『5.2 Linux カーネルの作成』で作成した Linux カーネルのフォルダに移動します。

```
省略 $ cd ~/xg3358-lk/linux-3.2.0-xg3358-X.X    入力
```

② 『**make ARCH=arm menuconfig**』コマンドによって、コンフィグレーションメニューを表示します。

```
省略 $ make ARCH=arm menuconfig    入力
scripts/kconfig/mconf Kconfig
```

③ メニューの『**Device Drivers**』－『**Input device support**』－『**Touchscreens**』と選択して、以下の画面の『**LCD-KIT-B01 based touchscreens**』もしくは『**LCD-KIT-C01 based touchscreens**』を選択します。



LCD-KIT-B01 と LCD-KIT-C01 の両方選択はしないでください。
必ず接続機器のみチェックをお願いします。
上記の画面で何も項目が表示されない場合は、一つ上の階層の『**Touchscreens**』がチェックされているかご確認ください。

8.3　動作確認

『**8.1　Linux カーネルの対応方法**』で作成したカーネルで起動した XG-3358 上で、タッチパネル LCD キット用のサンプルアプリケーションを動作させる手順を説明します。
なお、ルートファイルシステムは、『**5.3　ルートファイルシステムの作成**』で作成した ramfs ルートファイルシステムを使用した方法で説明します。

① 『**4.2　ブートローダの起動**』の手順に従って、U-Boot を起動します。
なお、XG-3358 ボードには、以下のようにタッチパネル LCD キットを接続します。
詳しい接続方法に関しては、使用するタッチパネル LCD キットの『**ハードウェアマニュアル**』でご確認ください。

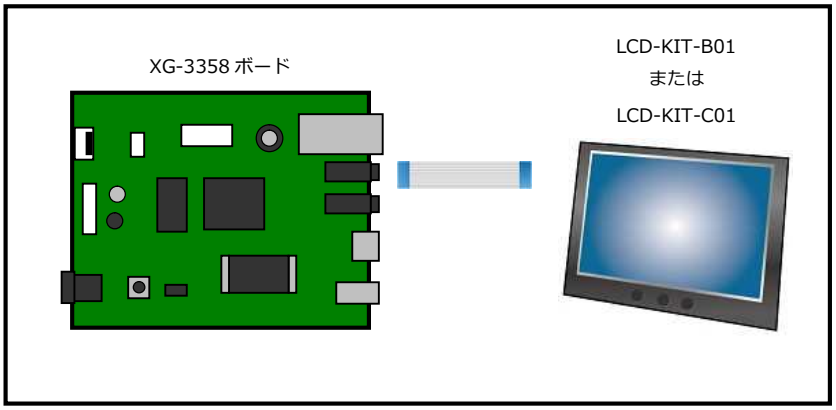


Fig 8.3-1　XG-3358 とタッチパネル LCD キットの接続

② 『**8.1　Linux カーネルの対応方法**』で作成した Linux カーネルイメージ『**uImage-xg3358-lcdkit**』を RAM 上にダウンロードします。

```
U-Boot# tftp 83000000 uImage-xg3358-lcdkit      入力
link up on port 0, speed 1000, full duplex
Using cpsw device
TFTP from server 192.168.128.210; our IP address is 192.168.128.200
Filename 'uImage-xg3358-lcdkit'.

        :
    途中省略
        :

done
Bytes transferred = 3891120 (3b5fb0 hex)
```

⑪ XG-3358 からゲスト OS の『**/nfs**』ディレクトリをマウントします。

```
# mount -t nfs -o nolock 192.168.128.201:/nfs /mnt/nfs
```
入力

⑫ アプリケーションを実行します。
タッチパネルをタッチするとその時のタッチパネル LCD キットから取得した値を画面上に表示します。
終了する場合は、タッチパネルのプッシュスイッチ(SW3)を押してください。

```
# /mnt/nfs/lcdkit
```
入力

```
   ~~~~~~~~~~~~~~~~~~~~~~~~~~| DirectFB 1.4.15 |~~~~~~~~~~~~~~~~~~~~~~~~~~
        (c) 2001-2010  The world wide DirectFB Open Source Community
        (c) 2000-2004  Convergence (integrated media) GmbH
      ------------------------------------------------------------------

(*) DirectFB/Core: Single Application Core. (2012-04-18 06:38)
(*) Direct/Memcpy: Using libc memcpy()

          :
       以降省略
          :
```

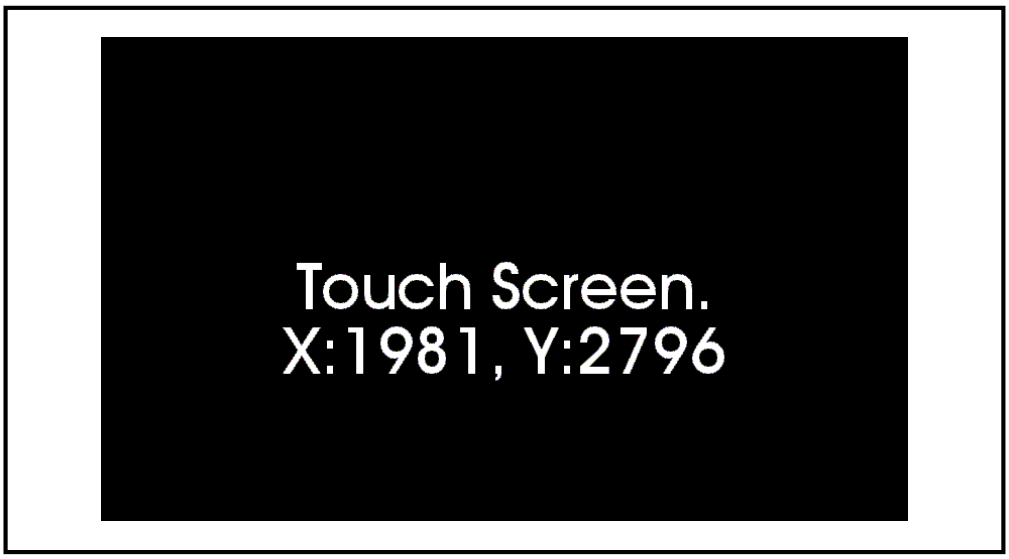


Fig 8.3-2　タッチパネル LCD キットのサンプル動作画面

表示している座標は、タッチパネル LCD キットのコマンドで取得した値そのままとなります。
値の詳細に関しては、『**LCD-KIT-B01 ハードウェアマニュアル**』もしくは『**LCD-KIT-C01 ハードウェアマニュアル**』でご確認ください。
なお、LCD-KIT-B01 はマルチタッチとなりますが、表示しているのは『**1st Finger touch**』のみとなります。

10. MMC ブート

XG-3358 は、MMC(microSD カード)ブートを行うことができます。
本手順では、MMC ブートを使用して、U-Boot の起動までの確認手順を説明します。

本手順では、Ubuntu 上で microSD カードへ U-Boot 等のデータをコピーする手順があります。
そのため、Ubuntu 上で認識できる SD カードリーダをご用意ください。

10.1 microSD カードの作成

作成の準備

① 作業用ディレクトリ『**xg3358-lk**』をホームディレクトリに作成します。
すでに作成されている場合は、手順②にお進みください。

```
省略 $ mkdir ~/xg3358-lk    入力
```

② 本手順で使用するデータの置く場所用にディレクトリ『**mmcboot**』を作業用ディレクトリに作成します。

```
省略 $ mkdir ~/xg3358-lk/mmcboot    入力
```

③ 手順②で作成したディレクトリに移動します。

```
省略 $ cd ~/xg3358-lk/mmcboot    入力
```

④ MMC ブートを使用して U-Boot を起動するためには、以下の 2 つのファイルが必要となります。

MLO
u-boot.img

『**4.4 MMC 起動用の U-Boot の作成**』で作成したファイルでも可能ですが、同等のファイルが CD 内にあります。
以下の手順⑤～⑦では、CD からコピーする方法で説明します。

⑤ CD をドライブに挿入します。
デフォルトでは、自動でマウントされますが、マウントされない場合は、以下のコマンドを実行します。

```
省略 $ gvfs-mount -d /dev/sr0    入力
```

マウントされているかどうかは、『**mount**』コマンドで確認できます。
以下のように、『**/dev/sr0**』が表示されている場合は、すでにマウントされています。
（『*******』は、CD のボリュームラベルになります。）

```
省略 $ mount    入力
          :
       途中省略
          :
/dev/sr0 on /media/******* type udf (ro,nosuid,nodev,uhelper=udisks,uid=1000,
gid=1000,iocharset=utf8,umask=0077)
```

④ パーティションを作成作業に入ります。
コマンドで手順③によって microSD カードの容量から計算されたシリンダ数を指定します。本手順の計算では、482 となりましたので、コマンドは、『**sudo sfdisk -D -H 255 -S 63 -C 482 /dev/sdb**』と入力して実行します。

```
省略 $ sudo sfdisk -D -H 255 -S 63 -C 482 /dev/sdb    入力
[sudo] password for guest:    入力
現在、誰もこのディスクを使っていないかを調べます...
OK

ディスク /dev/sdb: シリンダ数 482、ヘッド数 255、63 セクタ/トラック
古い場面:
警告: パーティションテーブルは C/H/S=*/49/48 として作成されたようです
  (482/255/63 のかわりに)。
このリストは、そのジオメトリと見なします。
ユニット = 1204224 バイトのシリンダ、1024 バイトのブロック、0 から数えます

   デバイス ブート 始点     終点 #シリンダ #ブロック   Id  システム
/dev/sdb1           3+   3292-   3290-   3868160    b  W95 FAT32
        開始: (c,h,s) 期待値 (3,23,33)  (1,2,3) を発見
        終点: (c,h,s) 期待値 (1023,48,48)  (960,48,48) を発見
/dev/sdb2           0       -       0          0    0  空
/dev/sdb3           0       -       0          0    0  空
/dev/sdb4           0       -       0          0    0  空
以下の書式で入力して下さい -- 指定しなかったフィールドには初期値をセットします
<start> <size> <type [E,S,L,X,hex]> <bootable [-,*]> <c,h,s> <c,h,s>
普通は <start> と <size> (そして恐らく <type>)を指定するだけで構いません。
```

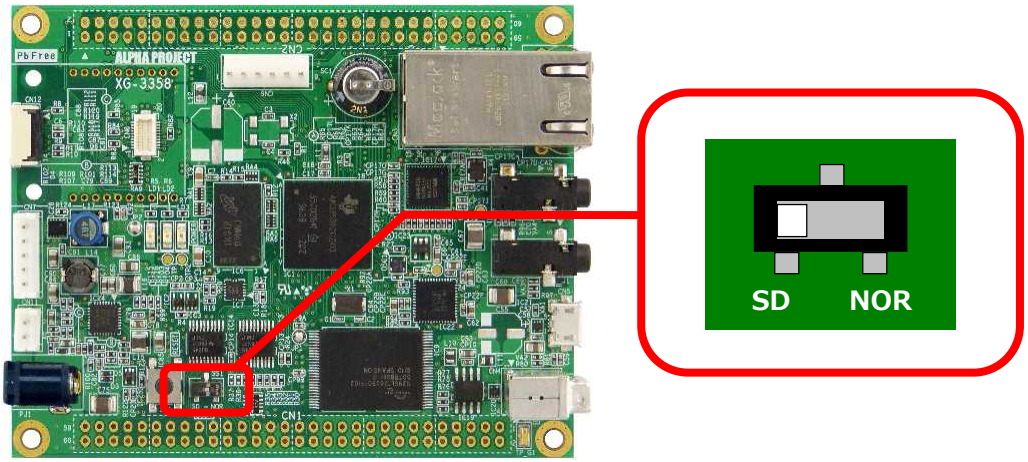
⑤ パーティションを作成します。パーティション 1 は、mmc ブートとして使用するため、『**,9,0x0C,***』と入力しますが、パーティション 2~4 は、何も入力しなくて Enter キーのみ押します。

```
/dev/sdb1 :,9,0x0C,*    入力
/dev/sdb1   *      0+      8       9-     72261    c  W95 FAT32 (LBA)
/dev/sdb2 :    入力    Enter キーのみ押します
/dev/sdb2          9     481     473-  3799372+  83  Linux
/dev/sdb3 :    入力    Enter キーのみ押します
/dev/sdb3          0       -       0          0    0  空
/dev/sdb4 :    入力    Enter キーのみ押します
/dev/sdb4          0       -       0          0    0  空
新たな場面:
ユニット = 8225280 バイトのシリンダ、1024 バイトのブロック、0 から数えます

   デバイス ブート 始点     終点 #シリンダ #ブロック   Id  システム
/dev/sdb1   *      0+      8       9-     72261    c  W95 FAT32 (LBA)
/dev/sdb2          9     481     473-  3799372+  83  Linux
/dev/sdb3          0       -       0          0    0  空
/dev/sdb4          0       -       0          0    0  空
```

10.2　起動手順

① XG-3358 のスイッチを以下の設定に変更します。
スイッチの設定の詳細に関しては、『**XG-3358 ハードウェアマニュアル**』でご確認ください。



② 下図に従って、『**10.1　microSD カードの作成**』で作成した microSD カードを XG-3358 に挿入し、PC-USB-04 を使用して PC と XG-3358 のシリアルポート（CN6）を接続します。

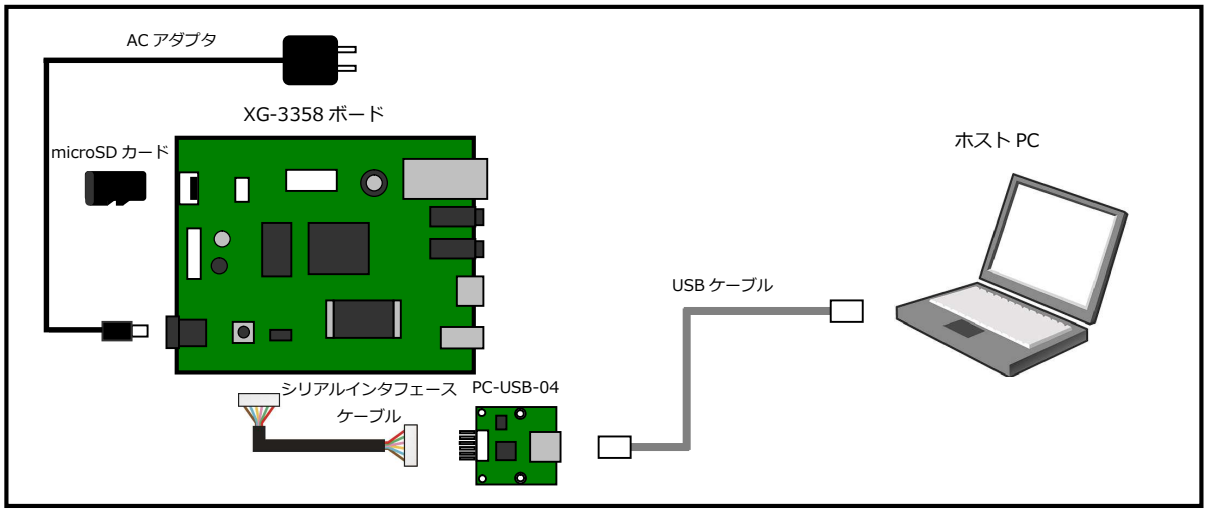


Fig 10.2-1 ボードの接続

PC-USB-04 とホスト PC を USB ケーブルで接続すると、PC-USB-04 がホスト PC に認識されて仮想 COM ポートが作成されます。
詳細に関しては、PC-USB-04 のマニュアルでご確認ください。

③ ホスト OS（Windows）のターミナルソフトを起動します。（設定は『**3.2　シリアル初期設定値**』を参照してください）

11. ボードの初期化

XG-3358 は、FlashROM に U-Boot、Linux カーネル等が書き込まれた状態で出荷しております。
再度 FlashROM へ U-Boot 及び Linux カーネル等を書き込みたい場合には、本章の手順に従って行ってください。

本手順では、MMC(microSD カード)ブートを使用して行いますので、Ubuntu 上で microSD カードへ Linux カーネル等のデータをコピーする必要があります。
そのため、Ubuntu 上で認識できる SD カードリーダをご用意ください。

11.1 FlashROM 構成

以下に、FlashROM のアドレスマップを記載します。
次節より、このアドレスマップになるように FlashROM へ書き込む方法を説明します。

FlashROM 32MByte

開始アドレス	領域名	領域サイズ
0x00000000	U-Boot	512KByte
0x00080000	U-Boot 環境変数領域	256KByte
0x000C0000	未使用	256KByte
0x00100000	Linux カーネル	3MByte
0x00400000	ramfs ルートファイルシステム	8MByte
0x00C00000	未使用(norflash ファイルシステム用)	20MByte

Fig 11.1-1 FlashROM 構成

11.2 作業概要

FlashROM に書き込む手順は、以下の 2 つの作業で行います。

1. MMC ブート用の microSD カードを作成します。
   microSD カードを MMC ブートできるように作成し、FlashROM に書き込むデータも入れます。

2. FlashROM に用意したデータを書き込みます。
   MMC ブートにより起動した U-Boot を利用して、FlashROM にデータを書き込む。

次節より、上記の作業順番で説明します。

④ パーティションを作成作業に入ります。
コマンドで手順③によって microSD カードの容量から計算されたシリンダ数を指定します。本手順の計算では、482 となりましたので、コマンドは、『**sudo sfdisk -D -H 255 -S 63 -C 482 /dev/sdb**』と入力して実行します。

```
省略 $ sudo sfdisk -D -H 255 -S 63 -C 482 /dev/sdb    入力
[sudo] password for guest:    入力
現在、誰もこのディスクを使っていないかを調べます...
OK

ディスク /dev/sdb: シリンダ数 482、ヘッド数 255、63 セクタ/トラック
古い場面:
警告: パーティションテーブルは C/H/S=*/49/48 として作成されたようです
  (482/255/63 のかわりに)。
このリストは、そのジオメトリと見なします。
ユニット = 1204224 バイトのシリンダ、1024 バイトのブロック、0 から数えます

   デバイス ブート 始点     終点 #シリンダ #ブロック   Id  システム
/dev/sdb1          3+   3292-   3290-   3868160    b  W95 FAT32
        開始: (c,h,s) 期待値 (3,23,33)  (1,2,3) を発見
        終点: (c,h,s) 期待値 (1023,48,48)  (960,48,48) を発見
/dev/sdb2          0       -       0          0    0  空
/dev/sdb3          0       -       0          0    0  空
/dev/sdb4          0       -       0          0    0  空
以下の書式で入力して下さい -- 指定しなかったフィールドには初期値をセットします
<start> <size> <type [E,S,L,X,hex]> <bootable [-,*]> <c,h,s> <c,h,s>
普通は <start> と <size> (そして恐らく <type>)を指定するだけで構いません。
```

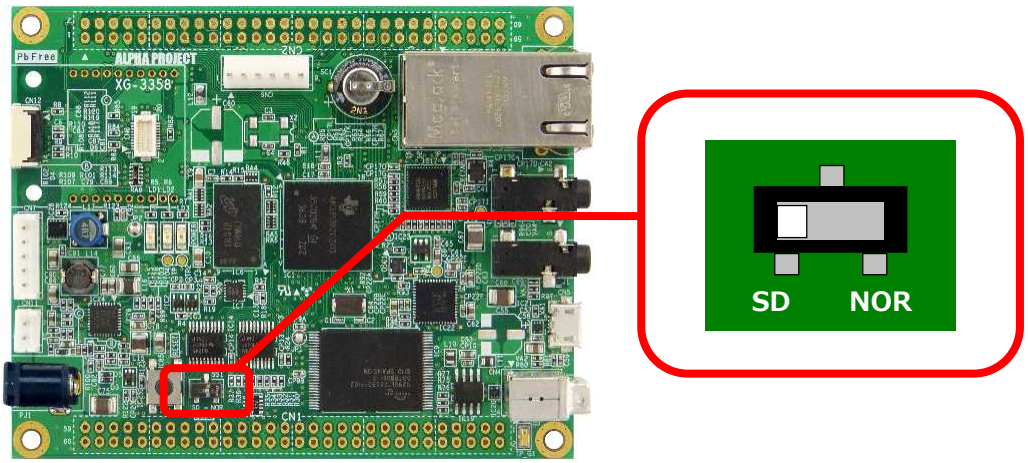
⑤ パーティションを作成します。パーティション 1 は、mmc ブートとして使用するため、『**,9,0x0C,***』と入力しますが、パーティション 2~4 は、何も入力しなくて Enter キーのみ押します。

```
/dev/sdb1 :,9,0x0C,*    入力
/dev/sdb1   *      0+      8       9-     72261    c  W95 FAT32 (LBA)
/dev/sdb2 :    入力    Enter キーのみ押します
/dev/sdb2          9     481     473    3799372+  83  Linux
/dev/sdb3 :    入力    Enter キーのみ押します
/dev/sdb3          0       -       0          0    0  空
/dev/sdb4 :    入力    Enter キーのみ押します
/dev/sdb4          0       -       0          0    0  空
新たな場面:
ユニット = 8225280 バイトのシリンダ、1024 バイトのブロック、0 から数えます

   デバイス ブート 始点     終点 #シリンダ #ブロック   Id  システム
/dev/sdb1   *      0+      8       9-     72261    c  W95 FAT32 (LBA)
/dev/sdb2          9     481     473    3799372+  83  Linux
/dev/sdb3          0       -       0          0    0  空
/dev/sdb4          0       -       0          0    0  空
```

11.4　書き込み手順

① XG-3358 のスイッチを以下の設定に変更します。
スイッチの各設定の詳細に関しては、『**XG-3358 ハードウェアマニュアル**』でご確認ください。



② 下図に従って、『**11.3　microSD カードの作成**』で作成した microSD カードを XG-3358 に挿入し、PC-USB-04 を使用して PC と XG-3358 のシリアルポート（CN6）を接続します。

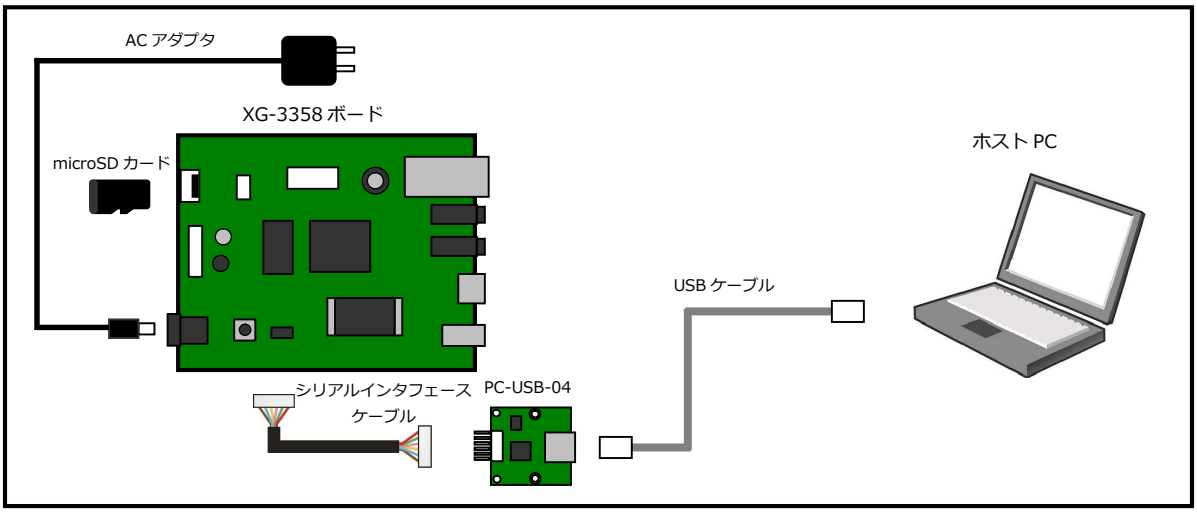


Fig 11.4-1 ボードの接続

PC-USB-04 とホスト PC を USB ケーブルで接続すると、PC-USB-04 がホスト PC に認識されて仮想 COM ポートが作成されます。
詳細に関しては、PC-USB-04 のマニュアルでご確認ください。

③ ホスト OS（Windows）のターミナルソフトを起動します。（設定は『**3.2　シリアル初期設定値**』を参照してください）

④ XG-3358 の電源を入れます。

⑤ ターミナルに、『**Hit any key to stop autoboot**』の文字が表示され、5 秒以内にキー入力を行うと U-Boot のコマンドコンソールが起動します。
コマンドコンソールが起動すると、『**U-Boot#**』が表示されます。

```
U-Boot SPL 2013.01.01 (Apr 21 2014 - 10:51:46)
musb-hdrc: ConfigData=0xde (UTMI-8, dyn FIFOs, bulk combine, bulk split, HB-ISO Rx, HB-ISO
Tx, SoftConn)
musb-hdrc: MHDRC RTL version 2.0
musb-hdrc: setup fifo_mode 4
musb-hdrc: 28/31 max ep, 16384/16384 memory

        :
    途中省略
        :

musb-hdrc: setup fifo_mode 4
musb-hdrc: 28/31 max ep, 16384/16384 memory
USB Host mode controller at 47401800 using PIO, IRQ 0
Net:   <ethaddr> not set. Validating first E-fuse MAC
cpsw
Hit any key to stop autoboot:  0     ← 入力   5 秒以内にキー入力を行います
U-Boot#
```

⑥ microSD カードを使用できるように準備します。

```
U-Boot# mmc rescan     ← 入力
```

⑦ microSD カード内の U-Boot を RAM にロードしてから、FlashROM に書き込みます。

```
U-Boot# fatload mmc 0 80000000 u-boot.bin     ← 入力
reading u-boot.bin
291244 bytes read in 33 ms (8.4 MiB/s)
U-Boot# protect off 08000000 +$filesize     ← 入力
... done
Un-Protected 3 sectors
U-Boot# erase 08000000 +$filesize     ← 入力

... done
Erased 3 sectors
U-Boot# cp.b 80000000 08000000 $filesize     ← 入力
Copy to Flash... done
```

謝辞

Linux、U-Boot の開発に関わった多くの貢献者に深い敬意と感謝の意を示します。

著作権について

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