ARM ボード-Mini2440 組込 Linux 移植ステップバイステップガイドライン

夢は実践から始まる

株式会社日昇テクノロジー

http://www.csun.co.jp

info@csun.co.jp

作成・更新日：2013/08/27

• 修正履歴

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2013/08/27

※ この文書の情報は、文書を改善するため、事前の通知なく変更されることがある。最新版は弊社ホームページからご参照ください。
「http://www.csun.co.jp」

目录

第一章　Git のインストールと使用（Fedora 9 に基づいて）

Git は Linux カーネル開発用のバージョン制御工具で、汎用のバージョン制御工具と違い、Git は分散リポジトリのバージョンライブラリの方式を採用し、サーバー側のソフトウェアサポートが必要なく、ソースコードの公表やと交流非常に便利となる。Git の高速反応速度は Linux kernel のような大なプロジェクトに対しては一番の優位性である。そして Git の最も優れたのはマージトラッキング（マージトレース）機能である。

カーネル開発チームは Git でカーネル開発のバージョン制御システムとして使うことを決める時、ワールド・オープンソース·コミュニティから反対声が多くある：Git が理解し難くすぎる。Git の内部動作メカニズムから見ればそうかもしれないが、開発と伴って、Git の正常使用はフレンドリーシナリオコマンドより簡単実行され、Git の使用は段々簡単となり、内部開発プロジェクトを管理するに使用されても、Git は強力なツールである。現在有名なプロジェクトは Git で項目開発を管理する、例えば wine、 U-boot など。http://www.kernel.org/git を参照する

1.1　Git ソースコードをダウンロード

ダウンロードアドレス：http://www.kernel.org/pub/software/scm/git/

本マニュアル作成時、最新バージョンは 1.6.6

1.2　Git をインストール

(1) #tar xvzf git-1.6.6.tar.gz

(2) #cd git-1.6.6

(3) #./configure --prefix=/usr/local

説明：インストールパス設定、git は/usr/local ディレクトリ下にインストールされる

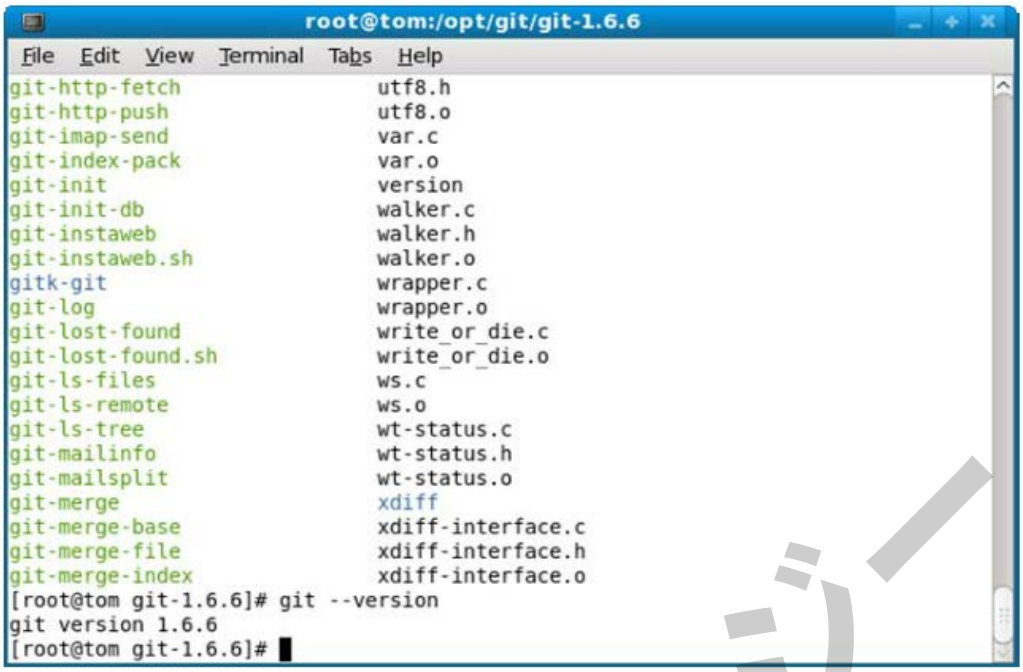
(4) #make

(5) #make install

1.3　ボージョン番号をチェック

#git -version

git 工具を説明する



第二章　supervivi と開発環境

2.1　supervivi について

2.1.1　最新の supervivi とパーティションテーブル

Supervivi は vivi に基づき改善した bootloader であり、使用は簡単、初心者向きのシステム・ダウンロード、システム書き込み、製品量産化など優位性がある。

システムソフトウェアをサポートするために、最新バージョン supervivi ダウンロード先 http://www.dragonwake.com/download/arm9-download/linux-2.6.32.2/linux-images.zip（解凍後、supervivi-128M が最新版）

本文作成時の最新バージョンは 0945 で、Linux-2.6.32.2 以後のソフトウェアの書き込みをサポートできる。

Supervivi は Linux システムにパーティション制御があり、Supervivi-0945-2K のパーティション情報は、次の通り：

Linux システム		
このパーティションテーブルはカーネル arch/arm/mach-s3c2440/mach-mini2440.c の中の nand flash パーティション情報と対応する		
オフセット(16 進数)	サイズ(16 進数)	ストレージの内容説明
0	40000	Supervivi/vboot/u-boot など
40000	20000	Linux 起動パラメータ
60000	500000	Kernel、ここでは 5M である
560000	rest	root パーティション
WindowsCE5/6 システム		
オフセット(16 進数)	サイズ(16 進数)	ストレージの内容説明

0	40000	Supervivi/nboot など
40000	20000	CE パラメータ
60000	200000	BootLogo、ここでは 2M、最大は 24 ビット 1024x768 トゥルーカラーイメージをサポートする
260000	rest	

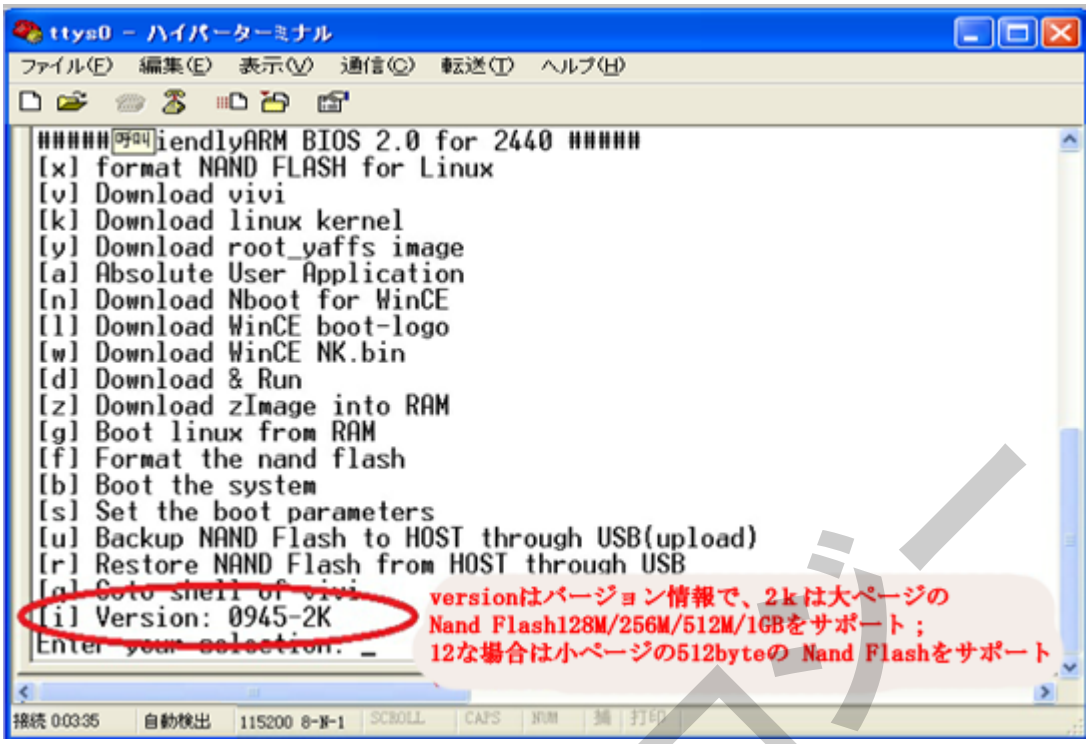
2.1.2 開発ボードの supervivi バージョンの識別方法

Supervivi が出荷時 NOR Flash はプリインストールされ、mini2440 開発ボードのスイッチ S2 は NOR 側に切り替えて、下図の通り通電・起動する。ボードの LED1 は点滅状態になりますそしてシリアルポートは下記の情報を出力する。supervivi が正常起動と確認できる。

注：supervivi モードでは LCD に出力がない、従って supervivi は開発ボードが LCD タイプを自動識別できず、LCD を初期化しない。

また、Supervivi は NAND Flash で起動する時、開発ボートの K1-K6 の任意ボタンを押すと、メニューに入る。

シリアルポートの出力は下図を参照。



Supervivi のシリアルポート出力情報[i] オプションで、supervivi のバージョンは 0945-2K と標識する。0945 は年間周期、2K は 2K ページの Nand Flash に適用と意味する。そして弊社生産の 128M/256M/1GB Nand Flash バージョンに適用する、バージョンは 0945-12 な場合、512 byte ページサイズの Nand Flash に適用する、64M Nand Flash バージョンにもサポートする。

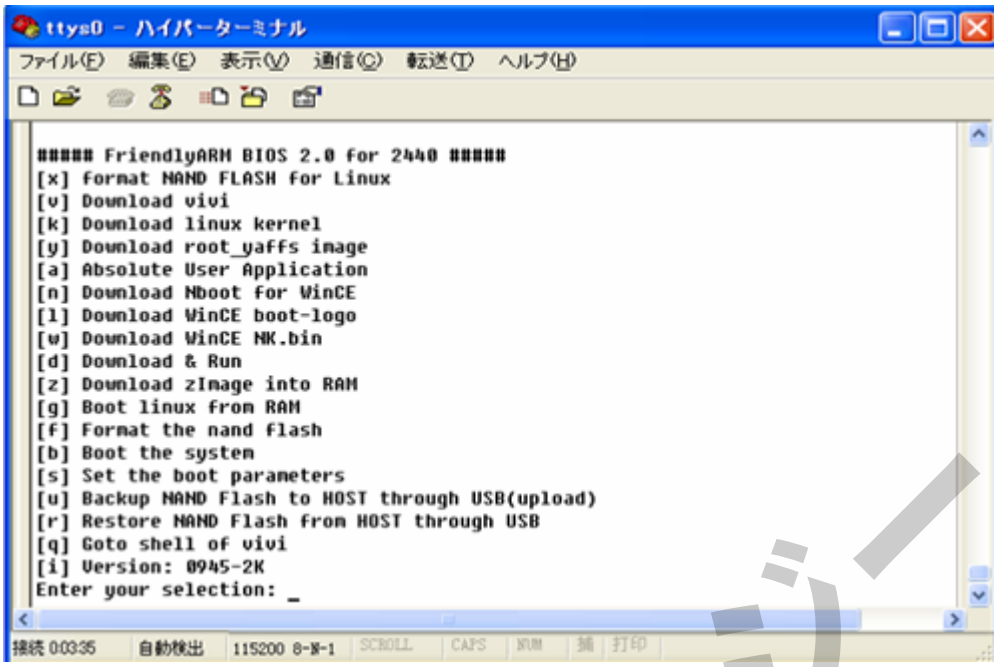
2.1.3 開発ボードの supervivi リカバリーとアップデート

誤操作で NOR Flash の supervivi ファイル破損、または update 出来ない場合、ユーザーマニュアル付録 2 を参照する

注：ソフトウェアを nand flash に書き込むため、NOR Flash 上の supervivi を最新バージョンに更新する必要がある。

2.1.4 supervivi 機能一覧

注：DNW プログラムを使用し、USB を通じてプログラムダウンロード機能を実現できる



機能[x]：Nand Flash にデフォルトのパーティションをし、Linux システムのみで有効。

機能［v]：USB を通じて Linux bootloader を Nand Flash の bootloader パーティションにダウンロードする

機能[k]：USB を通じて Linux カーネルを Nand Flash の kernel パーティションにダウンロードする

機能[y]：USB を通じて yaffs ファイル･システム･イメージを Nand Flash の root パーティションにダウンロードする

機能[a]：USB を通じてユーザープログラムを Nand Flash にダウンロードし、基本的にユーザープログラムは bin というような実行可能ファイルで、例えば 2440test(4K 以上の制限をサポートする必要がある)、uCos2(開発ボードにある uCos2 は nand flash 起動をサポートする)、U-Boot など；勿論、他の任意サイズの bin プログラムも実行可能。

機能[n]：USB を通じて WinCE の起動プログラム Nboot を Nand Flash の Block0 にダウンロードする

機能[l]：USB を通じて WinCE 起動の時のブート Logo(bmp フォームの画像)

機能[w]：USB を通じて WinCE 発行マッピング NK.bin を Nand Flash にダウンロードする

機能[d]：USB を通じてプログラムを指定しメモリアドレスにダウンロードする（DNW の Configuration->Option オプションを通じて実行アドレスを指定する)、実行する。本開発場オードに対して SDRAM の物理開始アドレスは 0x30000000 で、終了アドレスは 0x34000000 で、サイズは 64Mbytes で、その他 BIOS 自身は 0x33DE8000 以上のスペースを占めるため、BIOS の USB ダウンロード機能を使用する時、アドレスは 0x30000000 - 0x33DE8000 の間に指定される。

機能[z]: USB を通じて Linux カーネルイメージファイル zImage をメモリにダウンロードし、ダウンロードアドレスは 0x30008000 である。

機能[g]: メモリ上の Linux カーネルイメージを実行、機能［z]と合わせ使用される。

機能[f]：Nand Flash フォーマット、実行すると、Nand Flash のデータを全て消去する。(ユーザーズマニュアルの手順に従い初期化出来る)

機能［b]：システムを起動し、linux または wince をロードする時、コマンドを実行し、起動システムを自動識別出来る。

機能[s]：linux 起動パラメーターを設定し、サブメニューの説明を参照する

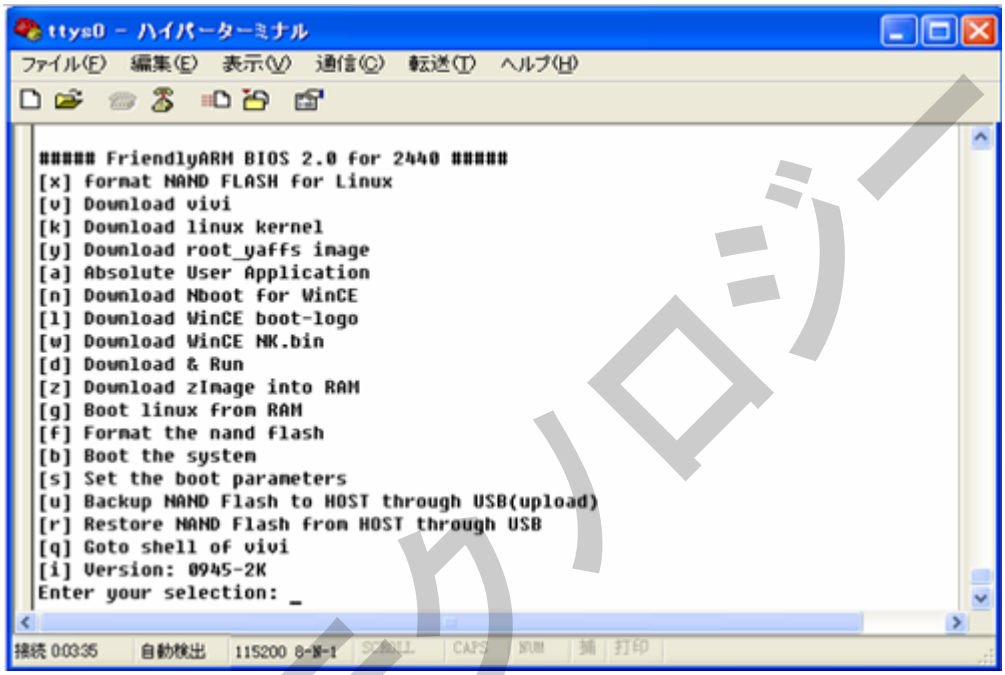
機能[u]：全体 Nand Flash の内容をバックアップし、USB を通じて PC にアップロードし、ファイルとして保存され、PC システムによく使われる Ghost ツールのようなものである。

功能[r]：バックアップしたファイルを使用し、Nand Flash に回復する

機能[i]：バージョン情報

機能[q]： vivi のコマンド shell モードに戻る。

下図を参照する



shell 画面で menu コマンドを入力すると、メニューに戻る

2.1.5 他のオープンソースの bootloader を使用

S3C2440 は NAND Flash 起動をサポートするため、開発ボードの S2 スイッチを NAND 側に切り替える場合、NOR Flash を OFF。本ボードは NOR Flash を保留し、Bootloader を NOR Flash に保存し、Nand Flash を管理（プログラミング/バックアップ）し、開発の効率も向上させる。開発中、Nand Flash の内容を破壊する場合もある、この時は JTAG ツールを使用し Nand Flash 中の Bootloader を初期化する必要がある。JTAG の手順は複雑且つ遅くて、NOR Flash の bootloader を通じて、supervivi は主に USB の高速ダウンロード特性を使用し、Nand Flash に対するプログラミング機能を実現する。

上記の supervivi 機能一覧表により、[a]機能を使用し、他の bootloader を Nand Flash の開始アドレス Block 0 に便利的にプログラミング出来る。他の bootloader は大きくない場合、例えばオープンソース U-Boot、または付属 DVD で提供した vboot や nboot など、[v]機能項目を使用し、プログラミングできる。結果は同じで、プログラムは Nand Flash の Block 0 開始アドレスにプログラミングする。

2.1.6 最新の supervivi について

(1) 最新の Supervivi は大容量カーネルパーティションをサポート、最大は 5M

(2) 最新の Supervivi は大寸法 WindowsCE 起動画像をサポート、最大に解像度 1024x768 の 24 ビットカ

ラー画像、または 280x800 解像度の 16 ビットトゥルーカラー画像をサポートする

（3）最新の Supervivi を使用し、最新の WindowsCE5/6 システムをプログラミングでき、起動時にシステムは一回のみ実行し、nand から起動する時は起動画面に止まる場合、supervivi のバージョンが古いのは原因だと思われる。

（4）最新の Supervivi は多種 Nand Flash タイプをサポートする

supervivi - 64M は 64M nand flash バージョンの開発ボードに適用し、具体的な nand flash モデルは：

- HY27US08121　韓国ヒュンダイ会社生産
- K9F1208 韓国サムスン会社生産

Supervivi - 128M は 128M/256M/512M/1GB nand flash バージョンの開発ボードに適用し、具体的なモデルは：

- K9F1G08: 128M
- K9F2G08 : 256M
- K9F4G08: 512M
- K9K8G08: 1GB

2.2　開発プラットフォームについて

Fedora 9 について

Fedora 9 は簡単なインストール/設定でき、root 権限を取得できる。（大部分の開発は root 権限に基づく）、Fedora 10 と以後のバージョンでは複雑な設定が必要とする。Linux の初心者では使い難くて、また Fedora 8 と以前のバージョンは機能が古い。従って、一番汎用性が高い Fedora 9 を使用する。マニュアルの手順に従い、Fedora 9 をインストールし、開発ソフトウェアパッケージを利用し、大半の機能を使用出来る。

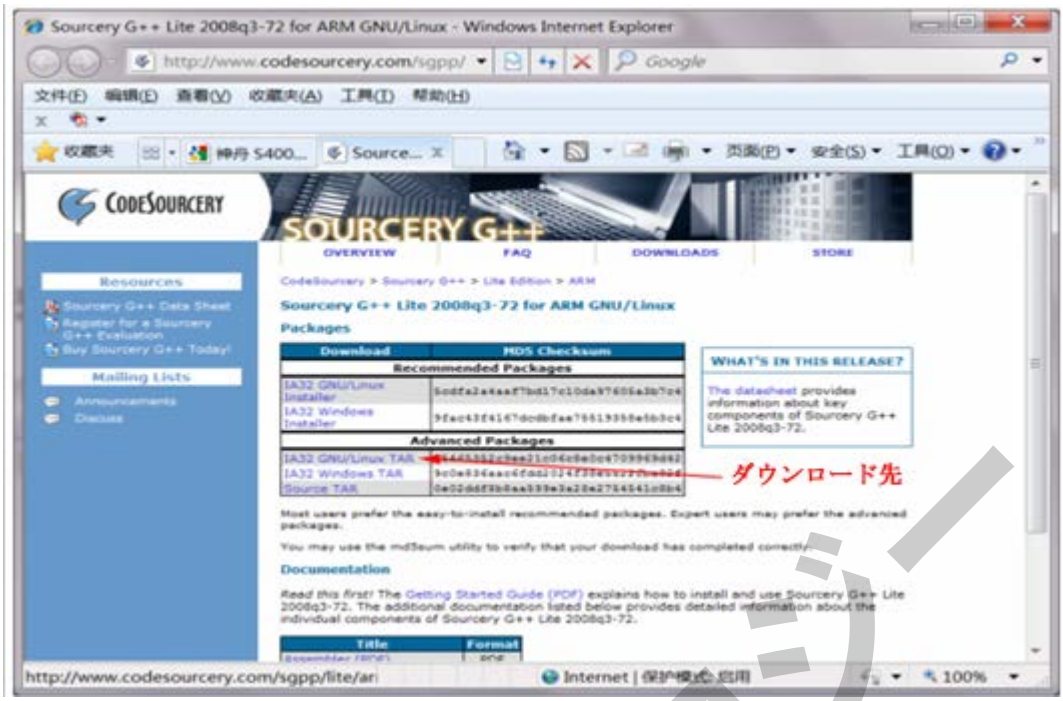
2.3　クロスコンパイラについて

2.3.1　取得先

クロスコンパイラ arm-linux-gcc 4.3.2 のダウンロード先：

http://www.codesourcery.com/sgpp/lite/arm/portal/release644

下図を参照する



ここでのバージョンは 2008q3 無料バージョンで、Lite Edition と呼ばれ、主にコマンドラインに基づいて、移植開発に対して十分である。ダウンロード先：

http://www.codesourcery.com/sgpp/lite/arm/portal/release1033

本マニュアルでは 2008q3 を使用する。

オープンソースソフトウェアを移植する時、他のライブラリを依存する場合もある。例えば png、zlib、jpeg など。これらの汎用ライブラリもコンパイラに移植し、付属 DVD のコンパイラのバージョンに統合した。

本コンパイラはクロスコンパイルで armv5 システムの構造のコマンドセットを生成し、使用する s3c2440 は ARM920T シリーズに所属し、ARMV4 のシステムの構造に基づき、armv4 コマンドセットをサポートするコンパイラを使用する。そのため、コンパイラ(主に gcc と g++)にシステムの構造パラメータを指定し、汎用の arm-linux-xxx 形式の実行可能なスクリプトにネーミングし、arm-linux-gcc のスクリプトは下記の通り：

```
#!/bin/bash

exec arm-none-linux-gnueabi-gcc -march=armv4t $*
```

2.3.2 ABI と EABI について

codesourcery 社が提供するコンパイラは EABI 標準のコンパイラで、古い ABI インターフェースコンパイラ、例えば arm-linux-gcc 2.95.3 などがコンパイルした実行可能のファイルは実行できない。

1、ABI とは

ABI、application binary interface (ABI) アプリケーションバイナリインタフェース

A。アプリケーション<ー>オペレーションシステム；

B。アプリケーション<ー> (アプリケーションが使用する) ライブラリ

C。アプリケーションの各コンポーネント間

API と類似の機能で、プログラム間のコードを統合させ、ABI の機能はプログラムをバイナリ（レベル）

互換性を実現する

2、OABI と EABI

OABI の O は〝Old〟、〝Lagacy〟、古い、OAB とは古い ABI である

EABI の E は〝Embedded〟、即ち新しい ABI である

EABI は GNU EABI

OABI と EABI は ARMCPU に通用する

3、EABI の優位性

A. ソフトウェア浮動小数点とハードウェアは浮動小数点機能の共用

B. システムの呼び出す効率はより高い

C. 今後のツールをサポート

D. ソフトウェア浮動小数点の場合、EABI のソフトウェア浮動小数点の効率は OABI より高い

4、OABI と EABI の区別

二つ ABI の区別は下記の通り：

A. call ルール（パラメータ伝送と戻り値取得の方法）

B. システムの呼び出す数とアプリケーションがシステムの読み出す方法

C. ターゲットファイルのバイナリフォーマット、プログラム·ライブラリーなど

D. システム構造のパディングと（padding/packing）アラインメント。

第三章　Linux-2.6.32.2 カーネル移植詳しい手順

3.1　序説

Linux-2.6.31 から Linux カーネルは mini2440 をサポートする。

Mini2440 のコア回路は SMDK2440 とほぼ同じで、Linux-2.6.32.2 カーネルは SMDK2440 に対するサポートは十分であるから、大部分の移植は出来る。ターゲットプラットフォームの差異により調整後、使用出来る。

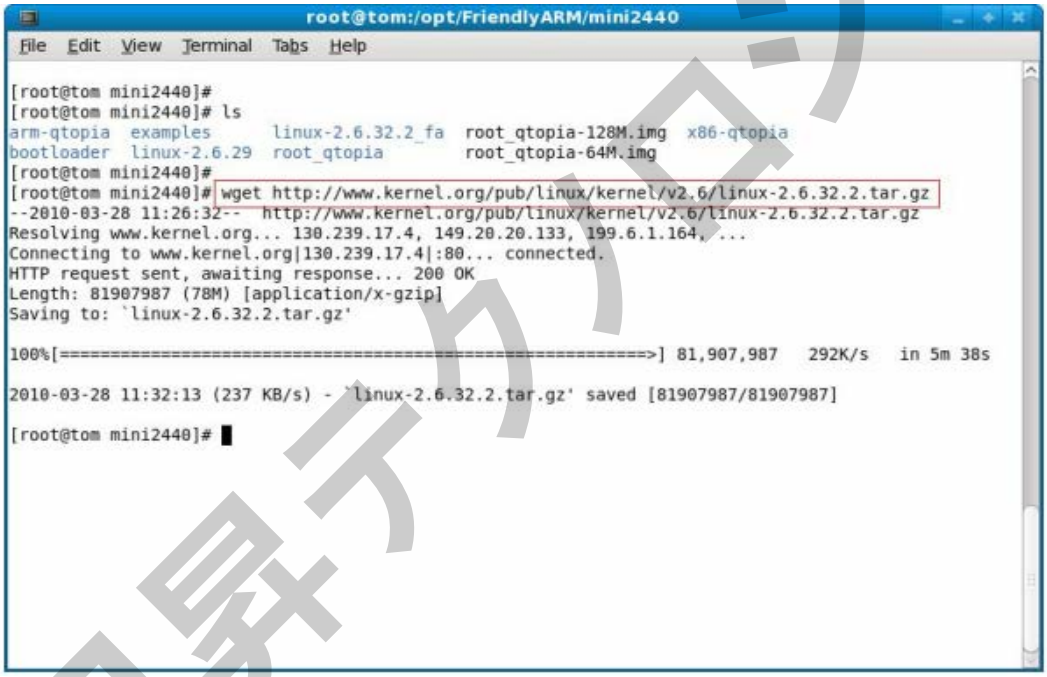
次は移植の詳しい説明である。

3.2　Linux カーネルソースコードを取得

Linux カーネルソースコードを取得する方法は多い、Fedora9 プラットフォームはネットと接続、直接にコマンドラインで下記のコマンドを入力直接ダウンロード出来る

#wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.2.tar.gz

また Windows システムでダウンロード後、Fedora9 にコピー出来る



3.3　カーネルソースコード解凍

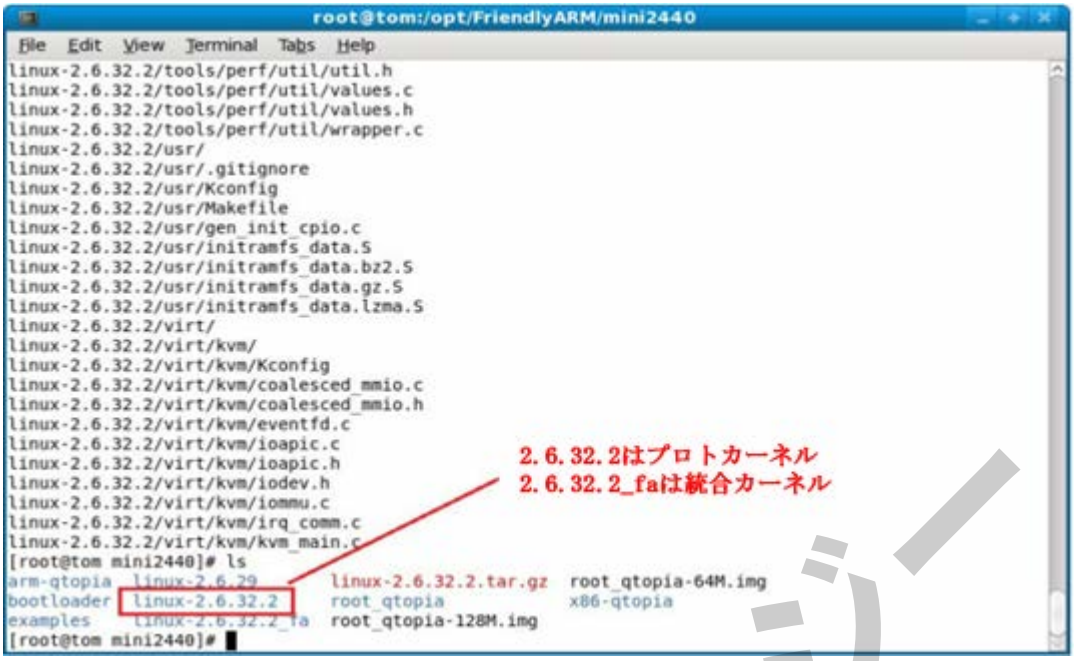
/opt/FriendlyARM/mini2440 ディレクトリにダウンロードしたカーネルソースコードを解凍する

コマンド：

#cd　/opt/FriendlyARM/mini2440

#tar　xvzf　linux-2.6.32.2.tar.gz

下図を参照する



3.4　クロスコンパイラの変数を指定

移植目的：Linux-2.6.32.2 は mini2440 で実行出来る。

まず Linux-2.6.32.2 のデフォルトターゲット·プラットフォームを ARM のプラットフォームに変更し、メインディレクトリの Makefile の

export KBUILD_BUILDHOST := $(SUBARCH)

ARCH ?= $(SUBARCH)

CROSS_COMPILE ?=

を次のように変更する

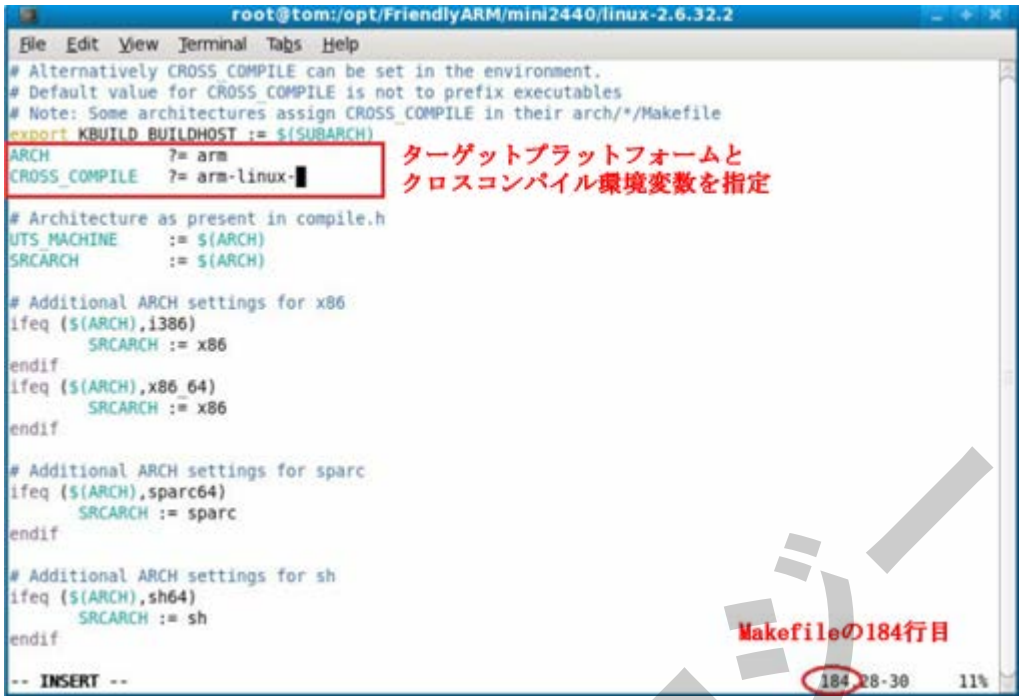
export KBUILD_BUILDHOST := $(SUBARCH)

ARCH ?= arm

CROSS_COMPILE ?= arm-linux-

ARCH はターゲットプラットフォームを arm と指定、CROSS_COMPILE はクロスコンパイラと指定する、フォルトとクロスコンパイラを指定後、他のものを使用する場合、コンパイラパスをここで書き込む。

注：vim エディタを薦め、特殊文字のカラー表示機能がある。下図を参照する



次は linux のコンパイルをテスト:

#make s3c2410_defconfig ； デフォルトのカーネルコンフィグレーションファイルを使用する、3c2410_defconfig は SMDK2440 のデフォルト設定ファイルである

#make ； コンパイル時間が長い

コンパイル成功

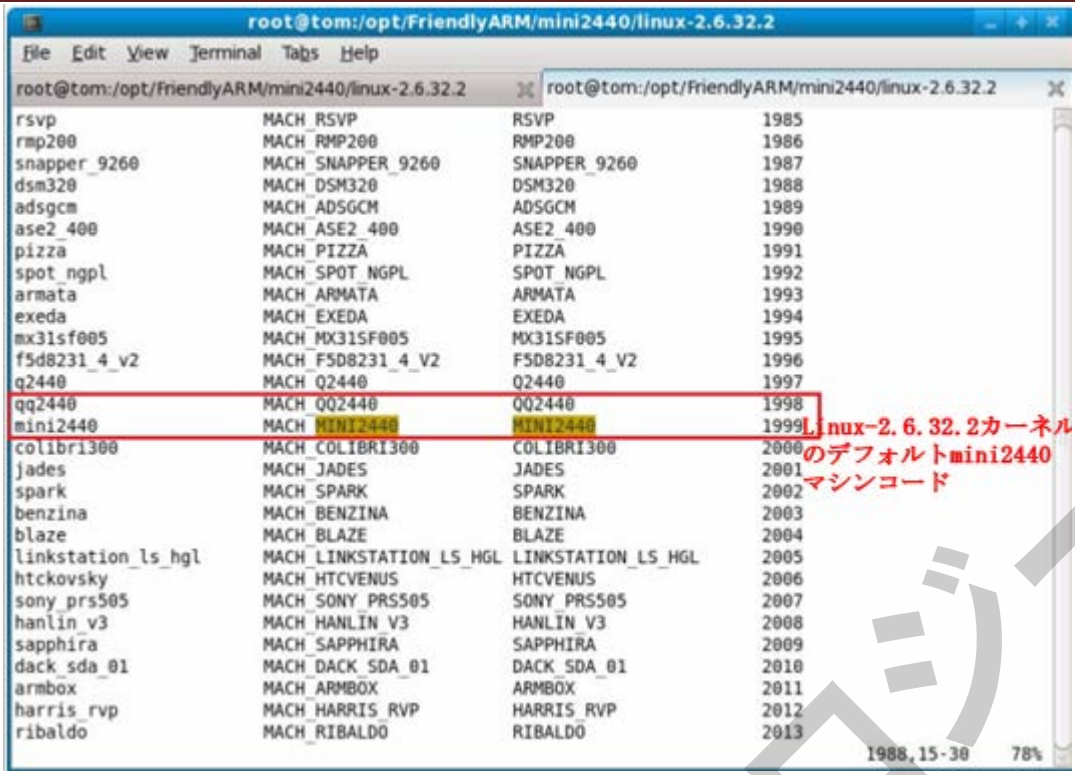
3.5 ターゲットプラットフォームを作成、クローン

3.5.1 マシンコードについて

上記のコンパイルは Linux カーネルがサポートするターゲットプラットフォーム設定を使用し、SMDK2440 と対応する。SMDK2440 を参照し、開発ボードにプラットフォーム MINI2440 を追加する。

Linux-2.6.32.2 は mini2440 をサポートし、名前は衝突した。ここで、やはり MINI2440 という名前を使用し、以後の移植手順にカーネル mini2440 のプロトコード部分を削除する。

まず、カーネルが起動する時、bootloader でマシンコード(MACH_TYPE)を転入、判断し、対応ターゲットプラットフォームで起動を指定する。本章で mini2440 のマシンコード 1999 で、ファイルは linux-2.6.32.2/arch/arm/tools/mach_types にある、下図を参照する



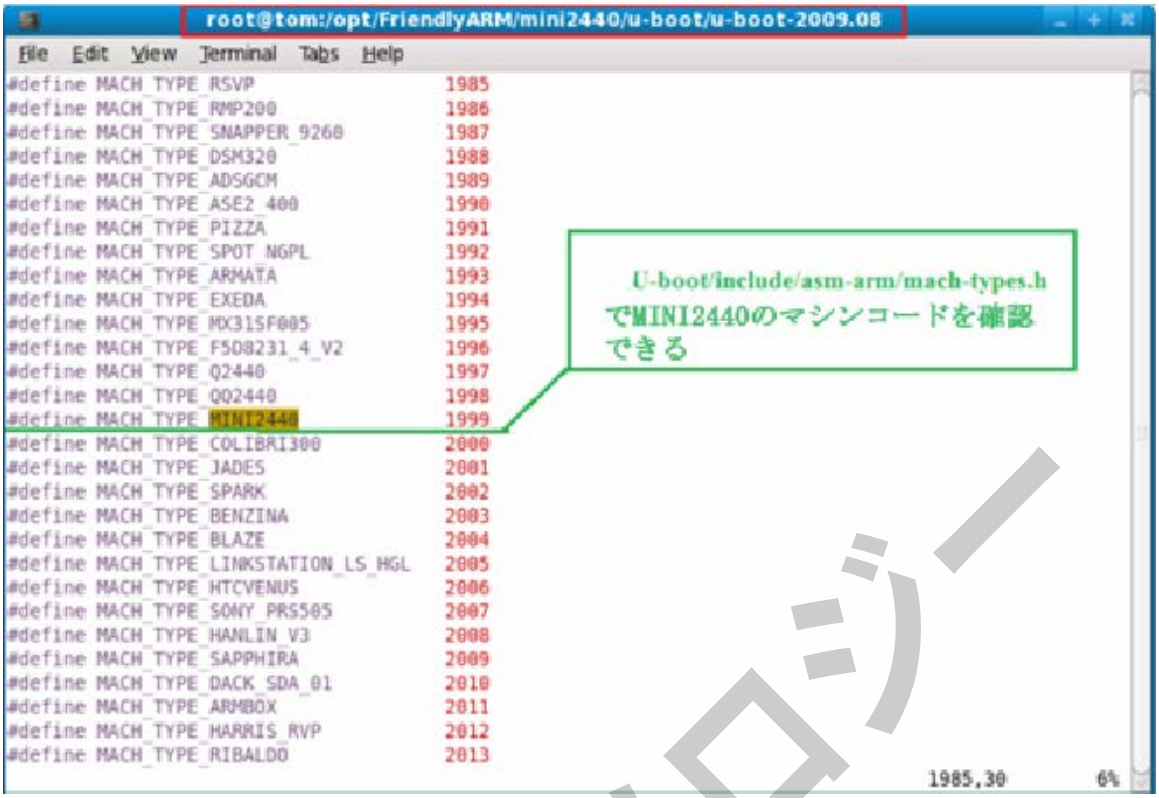
カーネルのマシンコードは bootloader のと一致しない場合、次のエラーが出る

```
Uncompressing Linux..... done,
booting
the kernel.
ここで止めます。
```

U-Boot の公式ウェブサイト(http://www.denx.de/wiki/U-Boot/WebHome)は 2009.06 バージョンから mini2440 のマシンコードの定義を追加した。他の方法で移植する u-boot は直接移植するカーネルを起動できる。

注：U-boot/include/asm-arm/mach-types.h で mini2440 のマシンコード定義を確認できる。

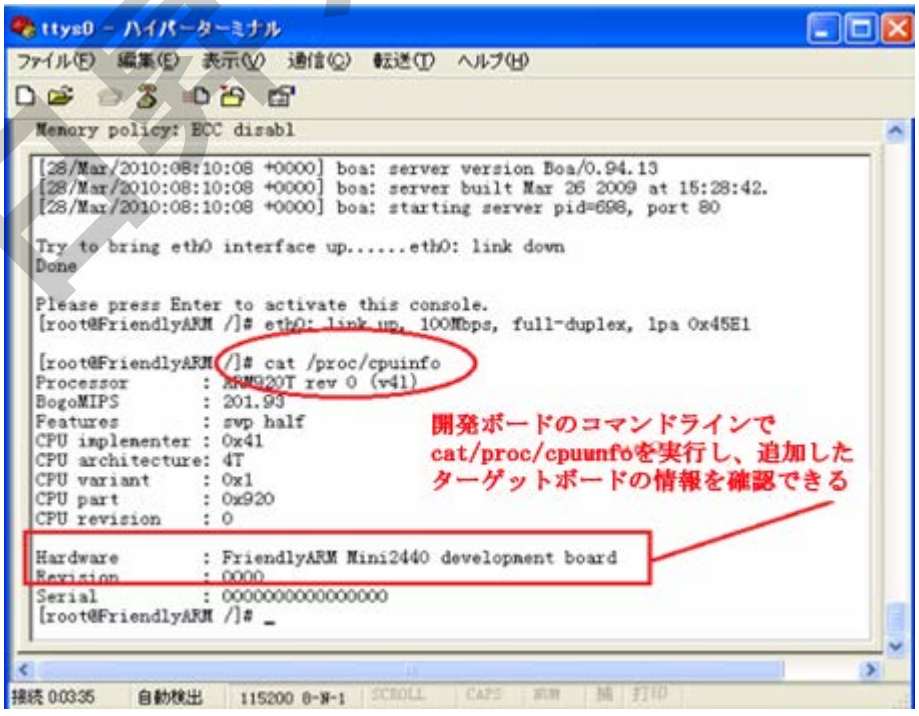
下図を参照する



次に、linux-2.6.32.2/arch/arm/mach-s3c2440 ディレクトリ下に mach-mini2440.c ファイルがある。これを使用しないため、直接削除する。

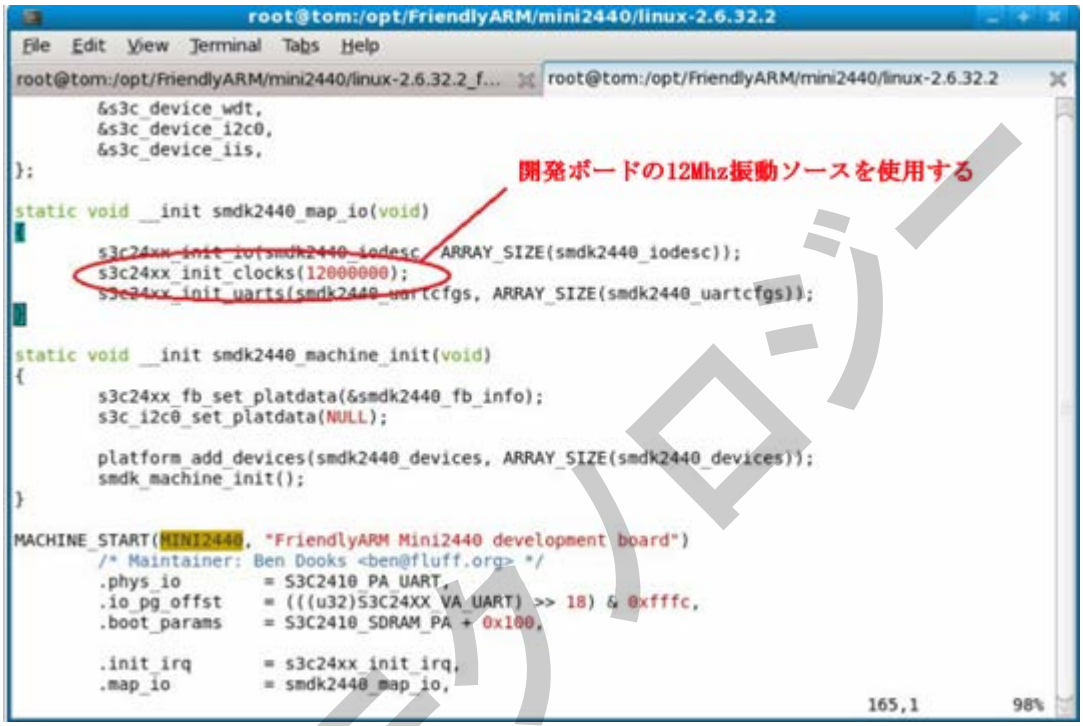
linux-2.6.32.2/arch/arm/mach-s3c2440/ディレクトリ下の mach-smdk2440.c をコピーし、mach-mini2440.c と名づけ、MACHINE_START(S3C2440、"SMDK2440")を検索、MACHINE_START(MINI2440、"FriendlyARM Mini2440 development board")に変更する。

注：開発ボード起動後、コマンドラインで：cat /proc/cpuinfo を入力する、開発ボード情報を確認できる：



3.5.2　クロックソース周波数の変更

システムクロックソースを変更する。mach-mini2440.c(mach-smdk2440.c をコピーしたもの)の第 160 行 static void __init smdk2440_map_io(void)関数で、中の 16934400(元の SMDK2440 目標ボードの水晶振動器 16.9344MHz を表示する)を mini2440 開発ボードに実際使用の 12000000(mini2440 開発ボードの水晶振動器 12MHz を表示し、コンポーネントのラベルは X2)に変更する、下図を参照する



3.5.3　SMDK2440 から MINI2440 まで

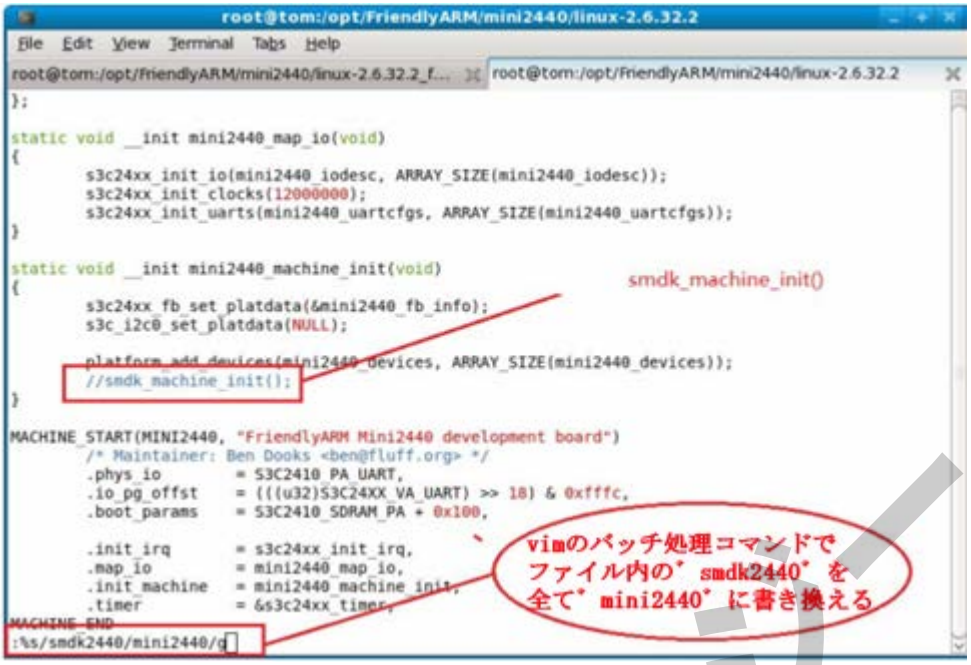
mini2440 プラットフォームシステムを作るので、区別するために、mach-mini2440.c 内のすべて smdk2440 字形は mini2440 に変更する、バッチ処理コマンドを使用できる。

vim のコマンドライン：

%s/smdk2440/mini2440/g

で変更する。

注：゛smdk2440゛とマッチングする文字列を全部゛mini2440゛に書き換える、前の゛%s゛は文字列マッチング、後の゛g゛は global/グロバル意味で、入力は下記の通り：



また、mini2440_machine_init(void)関数で、smdk_machine_init()関数のコメントを削除、smdk2440 のオリジナル関数は必要ない。上図を参照する。

3.5.4　コンパイルテスト

Linux ソースコードのルートディレクトリの下に実行する

#make mini2440_defconfig ； Linux 公式 mini2440 設定を使用する

#make zImage ；カーネル・コンパイル、長時間で、最後に zImage を生成する

再コンパイル、生成したカーネルファイル zImage (arch/arm/boot)はボードにダウンロードし、カーネルは正常に起動できる。そして大部分のハードウエアドライバ、ファイルシステムがインストールされていないため、登録できない。

```
##### FriendlyARM BIOS 2.0 for 2440 #####
[x] format NAND FLASH for Linux
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[a] Absolute User Application
[n] Download Nboot for WinCE
[l] Download WinCE boot-logo
[w] Download WinCE NK.bin
[d] Download & Run
```

[b] Boot the system
[s] Set the boot parameters
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
[i] Version: 0945-2K
Enter your selection: b
Copy linux kernel from 0x00060000 to 0x30008000, size = 0x00500000 ... done
zImage magic = 0x016f2818
Setup linux parameters at 0x30000100
linux command line is: "noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0"
MACH_TYPE = 1999
NOW, Booting Linux.....
Uncompressing
Linux.....
..... done, booting the kernel.
Linux version 2.6.32.2 (root@tom) (gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)) #3 Sun Mar 28 17:10:56
CST 2010
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
CPU: VIVT data cache, VIVT instruction cache
Machine: FriendlyARM Mini2440 development board
ATAG_INITRD is deprecated; please update your bootloader.
Memory policy: ECC disabled, Data cache writeback
CPU S3C2440A (id 0x32440001)
S3C24XX Clocks, (c) 2004 Simtec Electronics
S3C244X: core 405.000 MHz, memory 101.250 MHz, peripheral 50.625 MHz
CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256
Kernel command line: noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0
PID hash table entries: 256 (order: -2, 1024 bytes)
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 64MB = 64MB total
Memory: 60596KB available (3588K code, 417K data, 132K init, 0K highmem)
SLUB: Genslabs=11, HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1

再コンパイル、生成したカーネルファイルをボードにプログラミングし、〝b〟コマンドで起動すると、緑の起動情報が確認できます。そしてハードウエアドライバはまだインストールされません。システムは実行できません。

```
Hierarchical RCU implementation.
NR_IRQS:85
irq: clearing subpending status 00000003
irq: clearing subpending status 00000002
Console: colour dummy device 80x30
console [ttySAC0] enabled
Calibrating delay loop... 201.93 BogoMIPS (lpj=504832)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
NET: Registered protocol family 16
S3C2440: Initialising architecture
S3C2440: IRQ Support
S3C24XX DMA Driver, (c) 2003-2004,2006 Simtec Electronics
DMA channel 0 at c4808000, irq 33
DMA channel 1 at c4808040, irq 34
DMA channel 2 at c4808080, irq 35
DMA channel 3 at c48080c0, irq 36
S3C244X: Clock Support, DVS off
bio: create slab <bio-0> at 0
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
s3c-i2c s3c2440-i2c: slave address 0x10
s3c-i2c s3c2440-i2c: bus frequency set to 98 KHz
s3c-i2c s3c2440-i2c: i2c-0: S3C I2C adapter
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
```

```
JFFS2 version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
ROMFS MTD (C) 2007 Red Hat, Inc.
msgmni has been set to 118
alg: No test for stdrng (krng)
io scheduler noop registered
io scheduler anticipatory registered (default)
io scheduler deadline registered
io scheduler cfq registered
Console: switching to colour frame buffer device 60x53
fb0: s3c2410fb frame buffer device
s3c2440-uart.0: s3c2410_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2440
s3c2440-uart.1: s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2440
s3c2440-uart.2: s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2440
brd: module loaded
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
dm9000 Ethernet Driver, V1.31
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1
s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
usbcore: registered new interface driver libusual
mice: PS/2 mouse device common for all mice
S3C24XX RTC, (c) 2004,2006 Simtec Electronics
i2c /dev entries driver
S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics
s3c2410-wdt s3c2410-wdt: watchdog inactive, reset disabled, irq enabled
cpuidle: using governor ladder
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
usbcore: registered new interface driver hiddev
usbcore: registered new interface driver usbhid
usbhid: v2.6:USB HID core driver
```

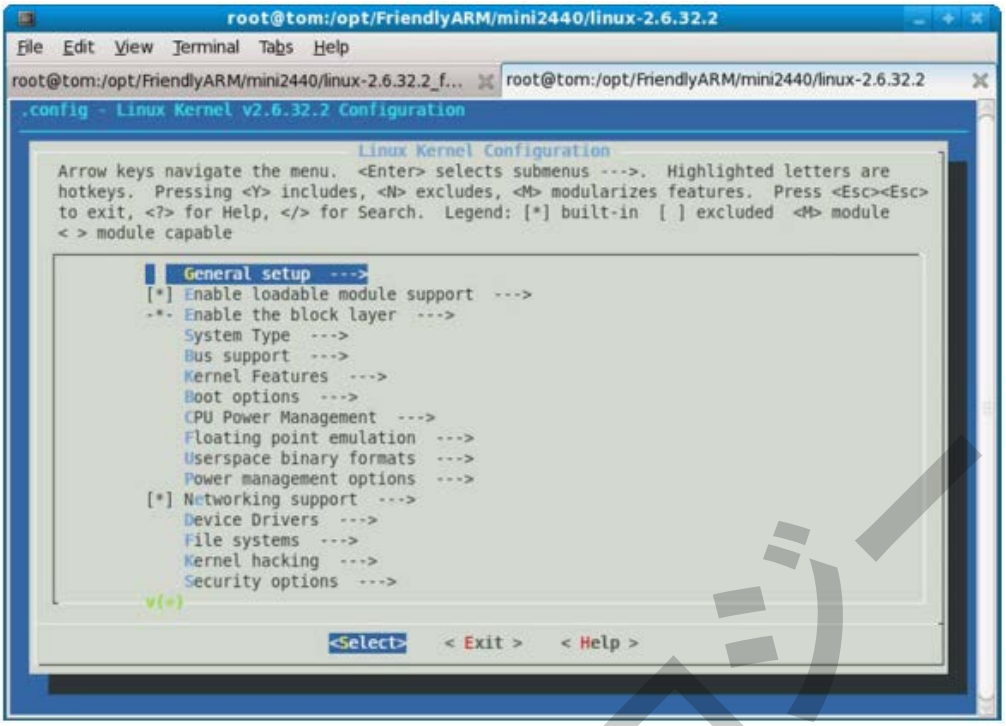
```
Advanced Linux Sound Architecture Driver Version 1.0.21.
No device for DAI UDA134X
No device for DAI s3c24xx-i2s
ALSA device list:
No soundcards found.
TCP cubic registered
NET: Registered protocol family 17
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
Root-NFS: No NFS server available, giving up.
VFS: Unable to mount root fs via NFS, trying floppy.
VFS: Cannot open root device "mtdblock3" or unknown-block(2,0)
Please append a correct "root=" boot option; here are the available partitions:
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(2,0)
[<c002e9f8>] (unwind_backtrace+0x0/0xdc) from [<c02d0390>] (panic+0x40/0x120)
[<c02d0390>] (panic+0x40/0x120) from [<c0008e84>] (mount_block_root+0x1d0/0x210)
[<c0008e84>](mount_block_root+0x1d0/0x210)from[<c000911c>]
prepare_namespace+0x164/0x1bc)
[<c000911c>] (prepare_namespace+0x164/0x1bc) from [<c000843c>] (kernel_init+0xd8/0x10c)
```

3.6　カーネル設定メニューの mini2440 オプションについて

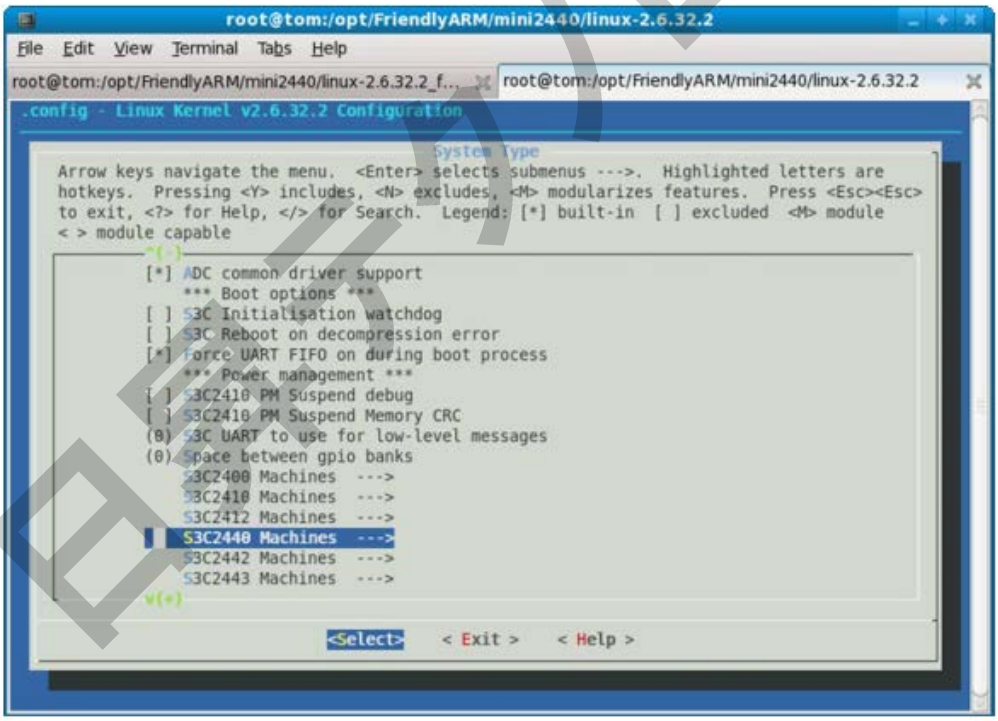
ドライバを移植する前に、make menuconfig 機能は必要である。カーネル設定メニューで mini2440 オプションを表示する：

#make menuconfig：前は既に make mini2440_defconfig を実行し、デフォルト設定がロードされるため、ここで該当コマンドを直接に実行できる。

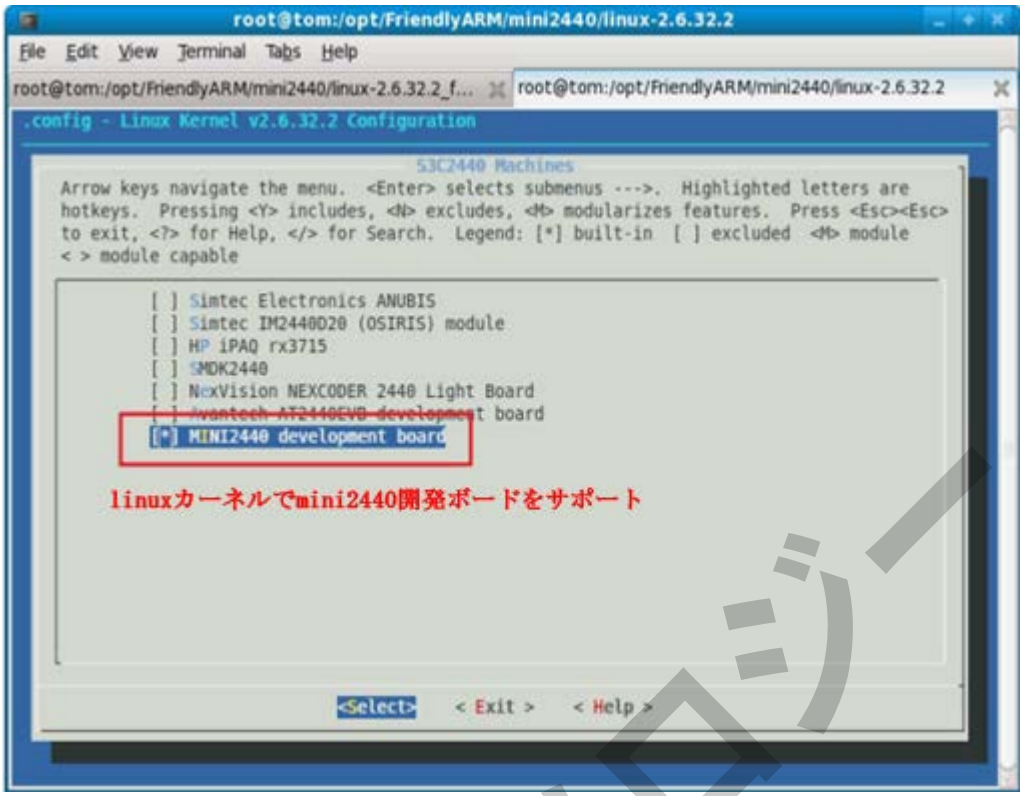
下図のようにカーネル設定メニューが出る



下キーで、System Type に移動し、Enter ボタンで、サブメニューに入る、下図を参照する

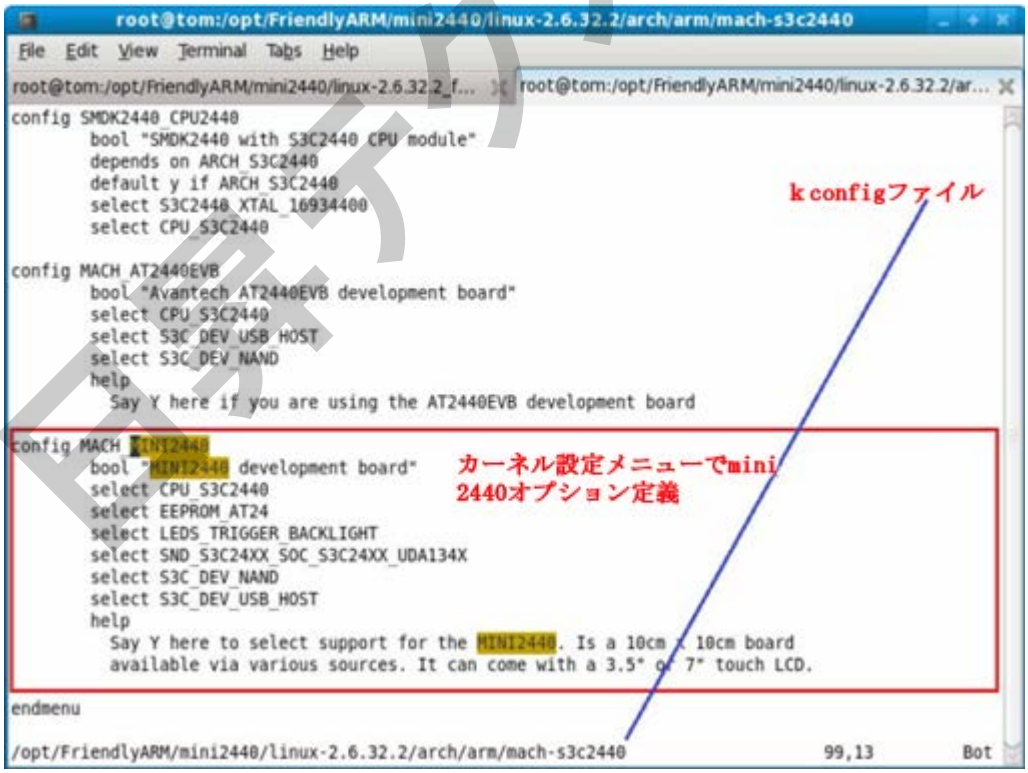


S3C2440 Machines で Enter ボタンを押し、サブメニューに入る、下図を参照する



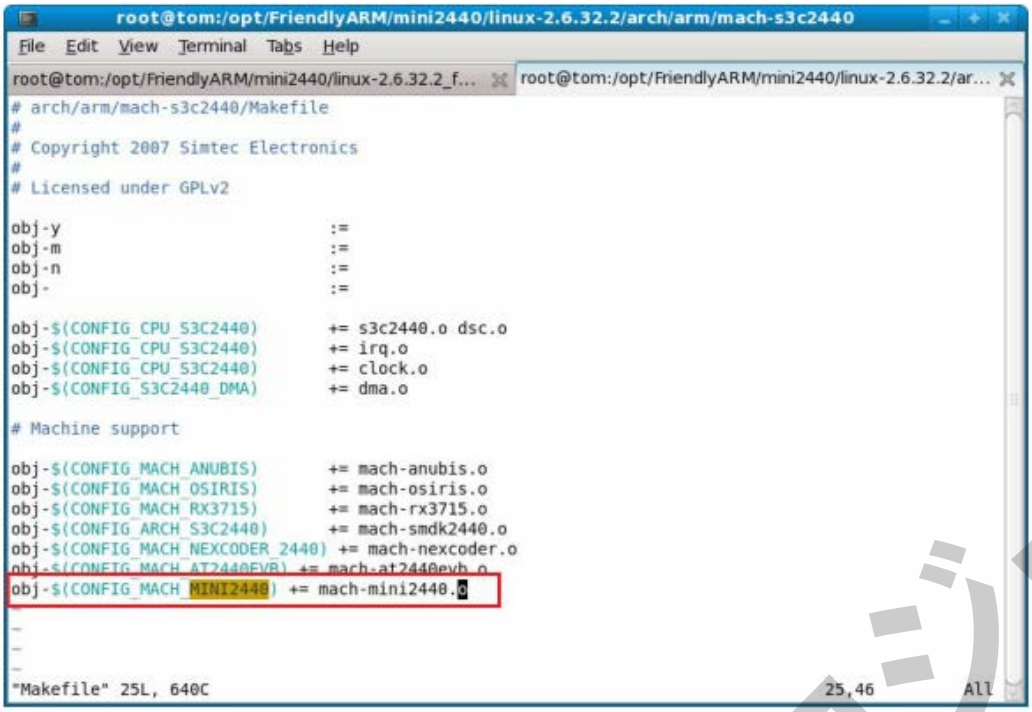
Linux のプロト・カーネルは mini2440 開発ボードに対するサポートオプションを選択。

Linux-2.6.32.2/arch/arm/mach-s3c2440/Kconfig ファイルをオープンすると、下図の情報を見つける



｀MINI2440 development board｀はこの Kconfig ファイルに定義され、表示情報を自由に変更できる。

ここの表示情報はカーネル設定メニューに表示するだけで、設定を有効きるまでは Makefile を使用し該当のコードファイルを追加する必要がある。該当ディレクトリの Makefile は下図を参照する。

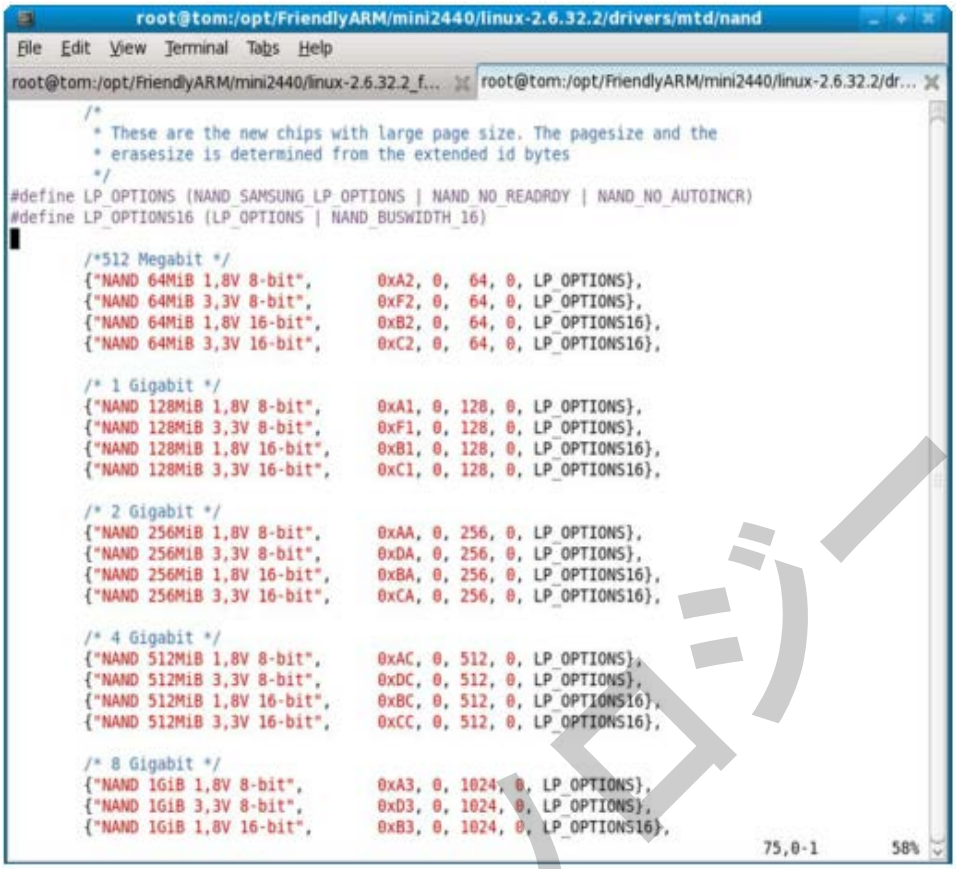


設定ファイルは実際のコードファイルと設定定義で繋がられる。ここの設定定義は`CONFIG_MACH_MINI2440`で、カーネルには類似設定定義があり、その一部の設定定義は依頼関係である。

3.7 Nand ドライバ移植、パーティション情報変更

3.7.1 Linux-2.6.32.2 カーネルはサポートする Nand Flash タイプ

Linux2.6.32.2 は既に殆どの Nand Flash ドライバを含め、linux-2.6.32.2/drivers/mtd/nand/nand_ids.c ファイルに、Nand Flash タイプを定義され、下図を参照する



3.7.2 Nand Flash パーティションテーブル変更

ここで、システムのデフォルトパーティションを自分で変更する必要がある。そして、システムの Nand Flash ドライバインタフェースに適合するように、Nand Flash の構造情報は書き込み必要がある。SMDK2440 に Nand Flash デバイスがレジスタされる情報を参照する

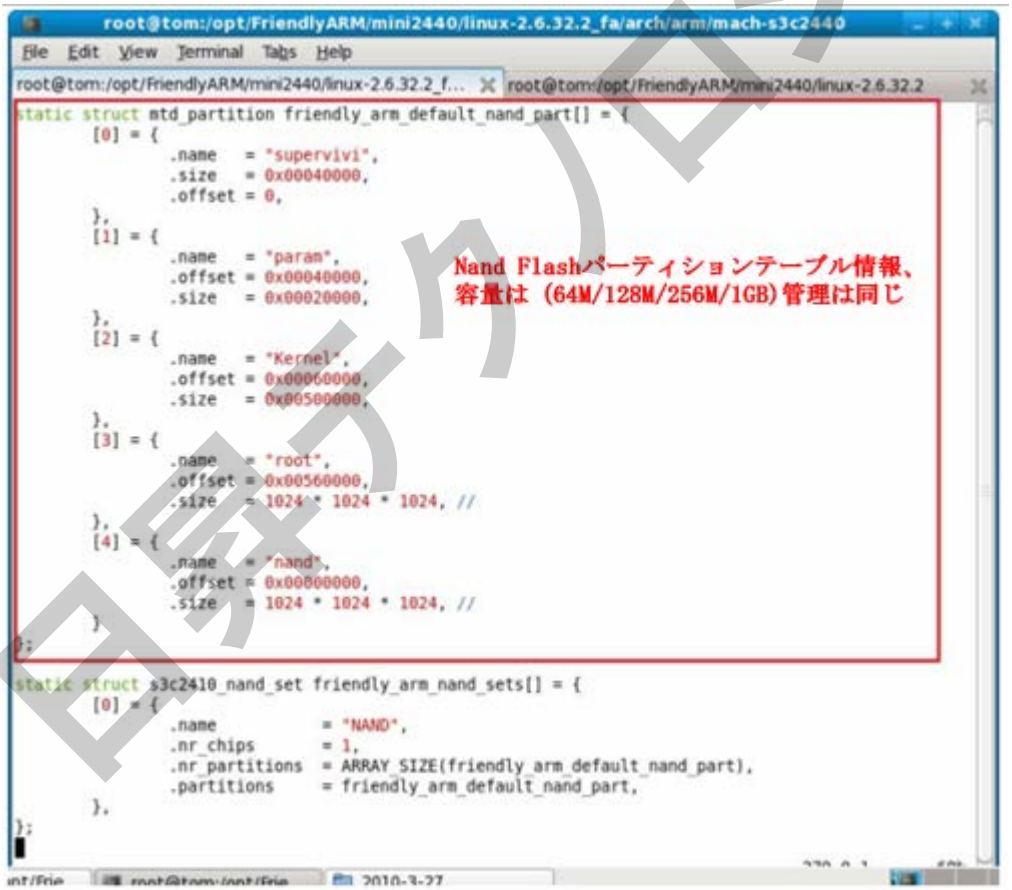
/arch/arm/plat-24xx/common-smdk.c をオープン：

```
static struct mtd_partition smdk_default_nand_part[] = {
        [0] = {
                .name   = "Boot Agent",
                .size   = SZ_16K,
                .offset = 0,
        },
        [1] = {
                .name   = "S3C2410 flash partition 1",
                .offset = 0,
                .size   = SZ_2M,
        },
        [2] = {
                .name   = "S3C2410 flash partition 2",
                .offset = SZ_4M,
                .size   = SZ_4M,
        },
        [3] = {
                .name   = "S3C2410 flash partition 3",
                .offset = SZ_8M,
                .size   = SZ_2M,
        },
        [4] = {
                .name   = "S3C2410 flash partition 4",
                .offset = SZ_1M * 10,
                .size   = SZ_4M,
        },
        [5] = {
                .name   = "S3C2410 flash partition 5",
                .offset = SZ_1M * 14,
                .size   = SZ_1M * 10,
        },
        [6] = {
                .name   = "S3C2410 flash partition 6",
                .offset = SZ_1M * 24,
                .size   = SZ_1M * 24,
        },
        [7] = {
                .name   = "S3C2410 flash partition 7",
                .offset = SZ_1M * 48,
                .size   = SZ_16M,
        }
};
```

上記は Nand Flash のパーティションテーブルで、Linux-2.6.32.2 に nand ドライバはプラットフォームデバイスとして登録され、/arch/arm/plat-24xx/common-smdk.c ファイルにも確認できる、次の通り

```
static struct s3c2410_platform_nand smdk_nand_info = {
        .tacls          = 20,
        .twrph0         = 60,
        .twrph1         = 20,
        .nr_sets        = ARRAY_SIZE(smdk_nand_sets),
        .sets           = smdk_nand_sets,
};

/* devices we initialise */

static struct platform_device __initdata *smdk_devs[] = {
        &s3c_device_nand,
        &smdk_led4,
        &smdk_led5,
        &smdk_led6,
        &smdk_led7,
};
```

上記の構造情報を参照し、mach-mini2440.c に内容を追加し、実現する。同時に Mini440 出荷時のカーネルのパーティションテーブルを参照する、下図を参照する



そして、mach-mini2440.c に次のコードを追加する。

```
青い文字はコメント
static struct mtd_partition mini2440_default_nand_part[] = {
[0] = {
                .name = "supervivi", ; bootloader のパーティションで、U-Boot、supervivi の内容を保存します。/dev/mtdblock0 と対応します
                .size = 0x00040000,
                .offset = 0,
        },
        [1] = {
                .name = "param", ; supervivi パラメータエリア、 bootloader の一部で、 u-boot のサイズが大きい場合、本エリアを上書きできます、システム起動には影響がありません、/dev/mtdblock1 と対応します
                .offset = 0x00040000,
                .size = 0x00020000,
        },
        [2] = {
                .name = "Kernel", ; カーネルのあるパーティションで、サイズは 5M、殆どの独自カスタムな巨大カーネルを保存できます、例えば巨大 Linux Logo など、/dev/mtdblock2 と対応します
                .offset = 0x00060000,
                .size = 0x00500000,
        },
        [3] = {
                .name = "root", ; ファイルシステムパーティション、ここでは yaffs2 ファイルシステムを保存します。/dev/mtdblock3 と対応します
                .offset = 0x00560000,
                .size = 1024 * 1024 * 1024, //
        },
        [4] = {
                .name = "nand", ;全体の nand flash を表します、予備用、例えば将来のプログラムが /dev/mtdblock4 を実行すると、nand flash 全体を バックアップできます。
                .offset = 0x00000000,
                .size = 1024 * 1024 * 1024, //
        }
};
; 開発ボード nand flash のセットアップ・テーブルです
```

```
static struct s3c2410_nand_set mini2440_nand_sets[] = {
        [0] = {
                .name = "NAND",
                .nr_chips = 1,
                .nr_partitions = ARRAY_SIZE(mini2440_default_nand_part),
                .partitions = mini2440_default_nand_part,
        },
};
;nand flash の特性で、 datasheet と対応して記入します、ほとんどの場合は次のパラメータを入力します
static struct s3c2410_platform_nand mini2440_nand_info = {
        .tacls = 20,
        .twrph0 = 60,
        .twrph1 = 20,
        .nr_sets = ARRAY_SIZE(mini2440_nand_sets),
        .sets = mini2440_nand_sets,
        .ignore_unset_ecc = 1,
};
```

nand flash デバイスをシステムに登録

```
static struct platform_device *mini2440_devices[] __initdata = {
        &s3c_device_usb,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c0,
        &s3c_device_iis,
        &s3c_device_nand, ;nand flash デバイスを開発ボードのデバイスリスト構造に加えます
};
```

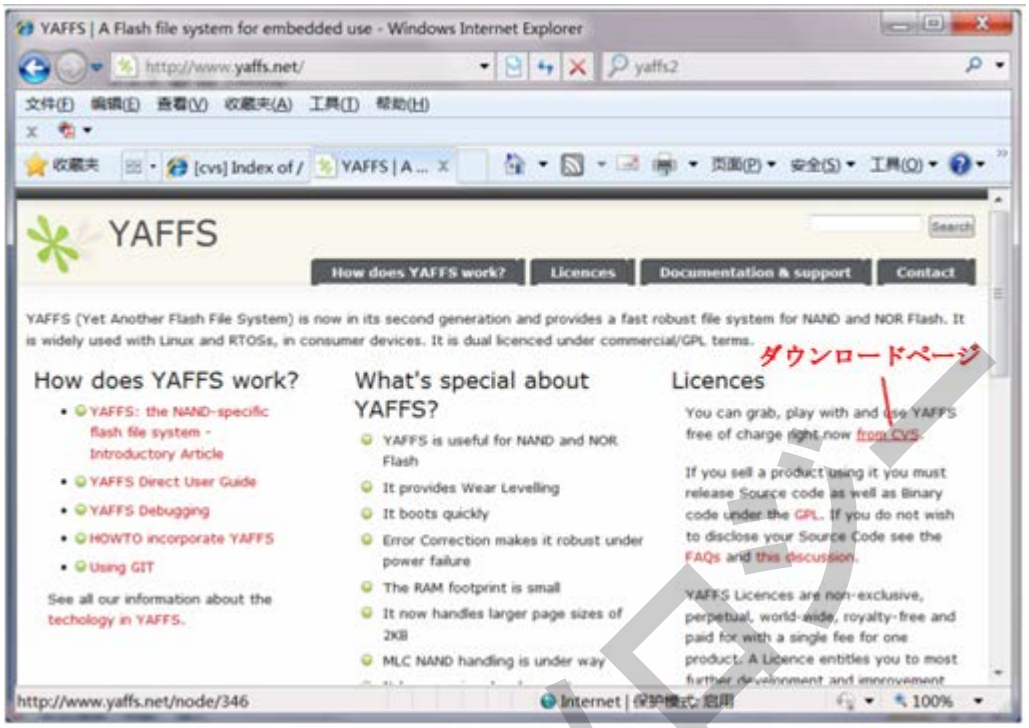
3.7.3 起動情報からパーティションテーブルを調べる

ここまで nand flash ドライバの移植は完了し、カーネルルート･ディレクトリで｀make zImage｀を実行し、生成する zImage を開発ボードにプログラミングし、起動後、赤字は先追加した Nand flash パーティション情報と開発ボード自身 nand flash の情報であり、ここでは 256M の nand flash である

```
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c24xx-nand s3c2440-nand: Tacls=3, 29ns Twrph0=7 69ns, Twrph1=3 29ns
s3c24xx-nand s3c2440-nand: NAND soft ECC
NAND device: Manufacturer ID: 0xec, Chip ID: 0xda (Samsung NAND 256MiB 3,3V 8-bit)
Scanning device for bad blocks
Bad eraseblock 329 at 0x000002920000
Bad eraseblock 399 at 0x0000031e0000
Bad eraseblock 878 at 0x000006dc0000
Bad eraseblock 982 at 0x000007ac0000
Bad eraseblock 1591 at 0x00000c6e0000
Creating 5 MTD partitions on "NAND 256MiB 3,3V 8-bit":
0x000000000000-0x000000040000 : "supervivi"
0x000000040000-0x000000060000 : "param"
ftl_cs: FTL header not found.
0x000000060000-0x000000560000 : "Kernel"
uncorrectable error :
0x000000560000-0x000040560000 : "root"
mtd: partition "root" extends beyond the end of device "NAND 256MiB 3,3V 8-bit" -- size truncated to 0xfaa0000
ftl_cs: FTL header not found.
0x000000000000-0x000040000000 : "nand"
mtd: partition "nand" extends beyond the end of device "NAND 256MiB 3,3V 8-bit" -- size truncated to 0x10000000
dm9000 Ethernet Driver, V1.31
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1
s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
```

3.8　yaffs2 を移植

3.8.1　yaffs2 ソースコードを取得



YAFFS の公式サイト

http://www.yaffs.net/node/346 で最新ソースコードをダウンロードする。git ツールを使用し（インストール方法は本マニュアルの第一章を参照する）、コマンド：#git　clone git://www.aleph1.co.uk/yaffs2 を実行し、最新の yaffs2 ソースディレクトリをダウンロード、また付属 DVD にソースコードパッケージもある、（ファイル：yaffs2-src-YYYYMMDD.tar.gz）

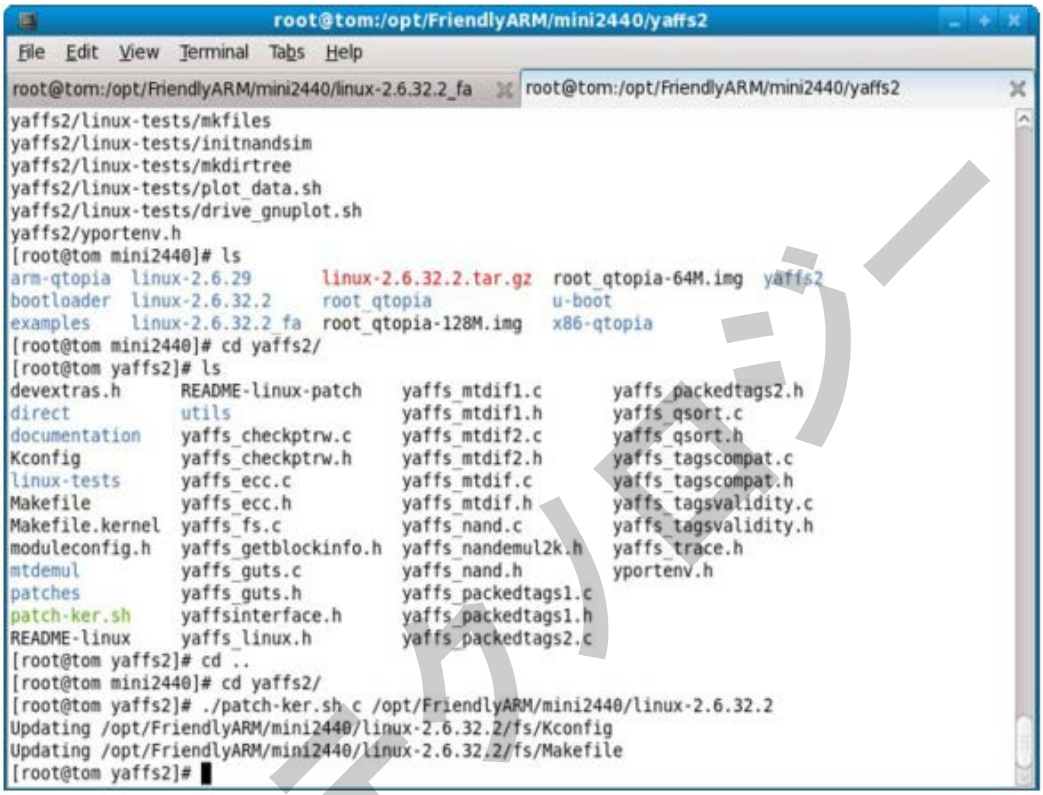
3.8.2　カーネルに yaffs2 パッチを追加

yaffs2 ソースコードディレクトリに入り、

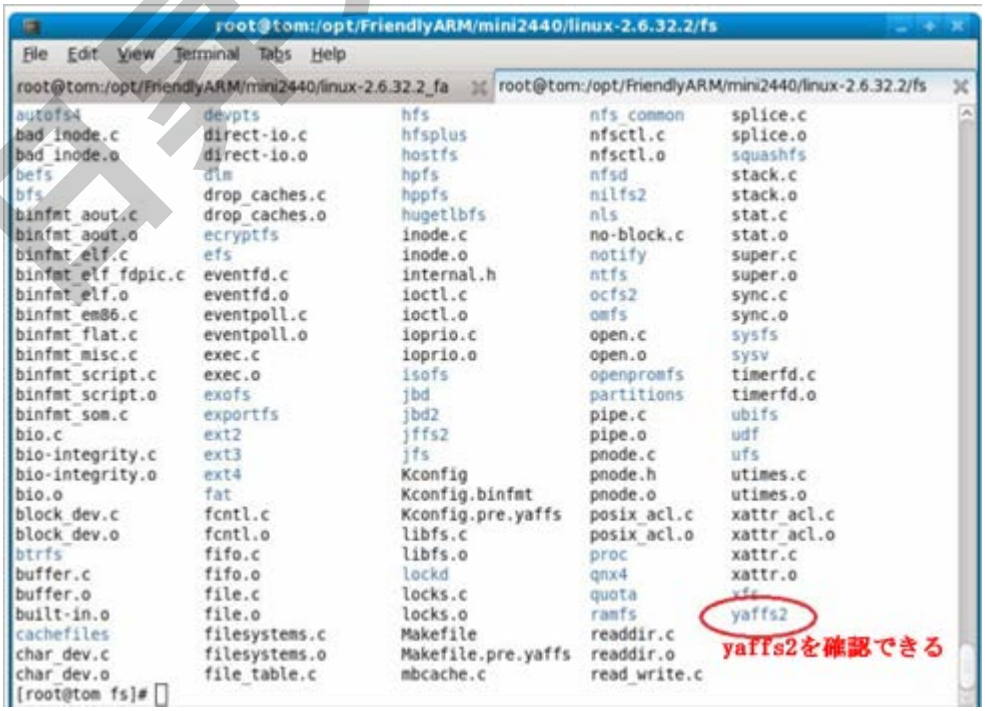
#cd yaffs2

#./patch-ker.sh c /opt/FriendlyARM/mini2440/linux-2.6.32.2 を実行し

下図の通り、yaffs2 パッチは成功に追加される

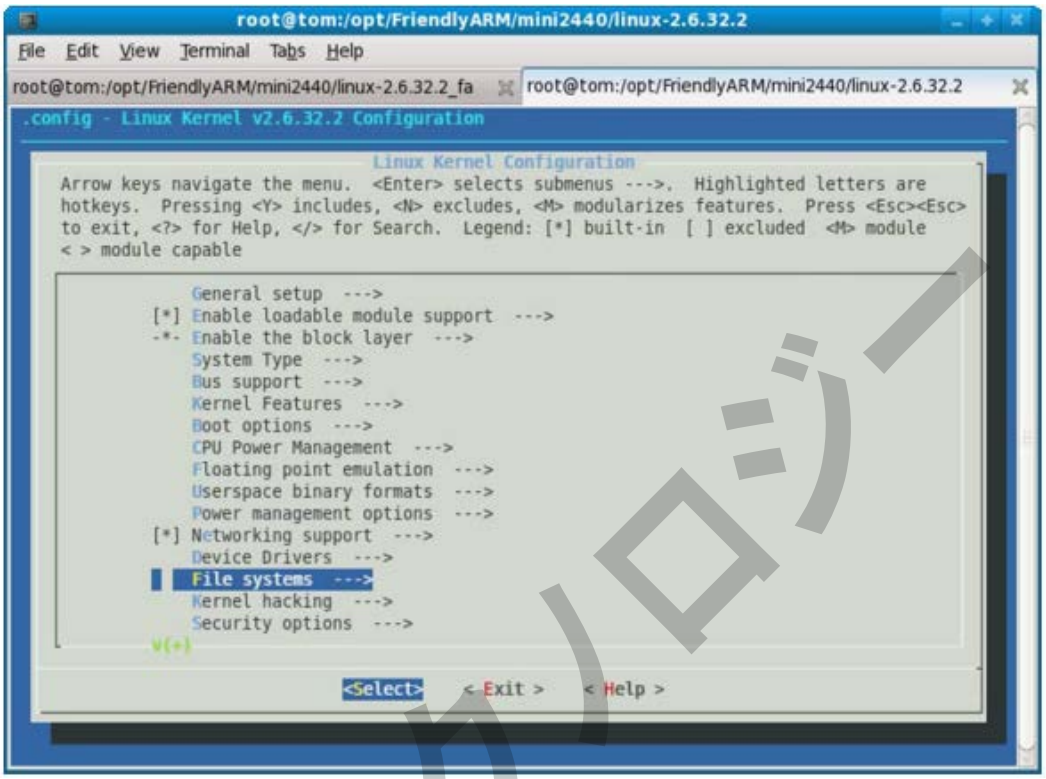


linux-2.6.32.2/fs ディレクトリに入り、yaffs2 ディレクトリを確認出来る：

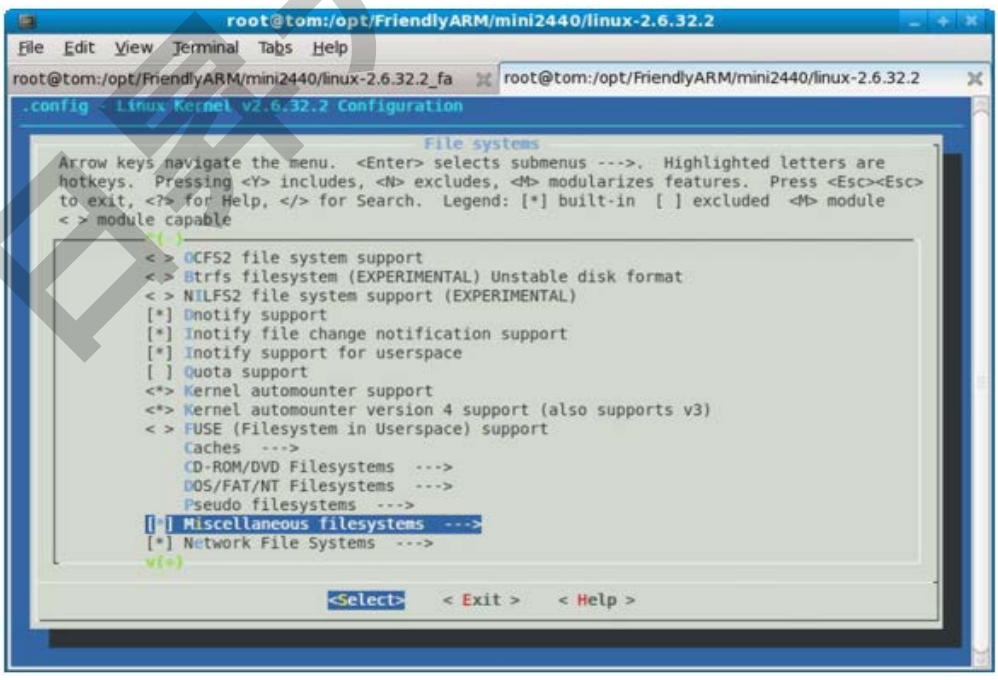


3.8.3　YAFFS2 をサーポートするカーネルをコンフィグ、コンパイル

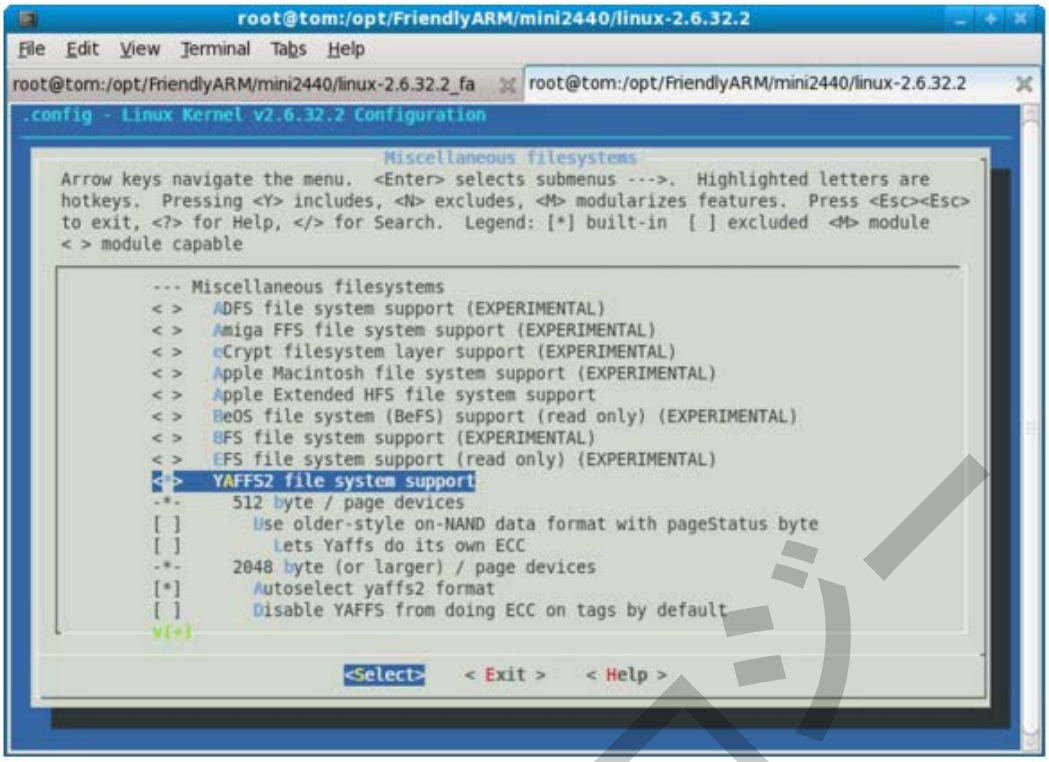
Linux カーネルソースコードルートディレクトリで make menuconfig を実行する。下向きボタンで移動し、File Systems を選べ Enter ボタンを押し、サブメニューに入る



〝Miscellaneous filesystems〞メニューオプションを探し、Enter ボタンを押し、サブメニューに入る、下図を参照する



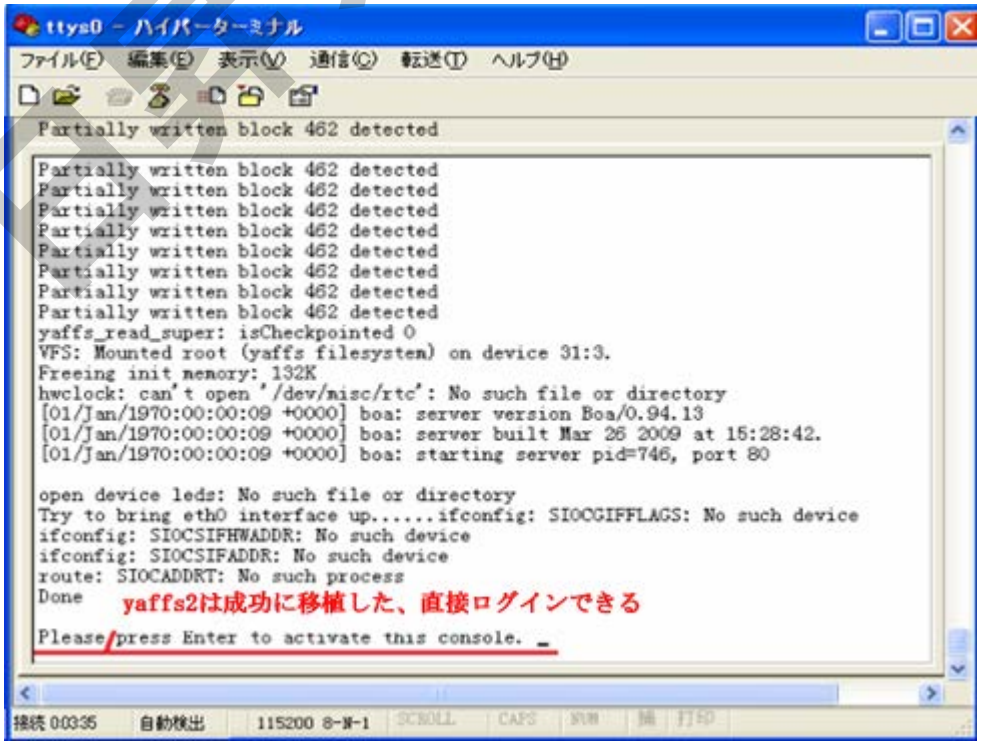
下図のメニューで、〝YAFFS2 file system support〞を探し、スペースボタンで選択する。カーネルに yaffs2 ファイルシステムのサポートを追加、〝Exit〞を押し、カーネル設定を終了する。



コマンドライン：#make zImage を実行する

3.8.4 開発ボードにプログラミング、実行、テスト

最後に linux-2.6.32.2/arch/arm/boot/zImage を生成し、supervivi の゛k゛機能を利用し、nand flash にプログラミングする、゛b゛を押し、システムを起動する。nand flash にはファイルシステムがあれば、（supervivi の゛y゛機能を使用し、既存の yaffs2 ファイルシステムイメージ root_qtopia-128M.img でテスト）。下図の情報を確認できましたら、yaffs2 の移植は成功することを明らかにしている。



3.9 DM9000 の NIC ドライバを移植

3.9.1 デバイスリソースの初期化

Linux-2..6.32.2 はすでに完全な DM9000 NIC ドライバを持っていた（ソースコード位置：linux-2.6.32.2/drivers/net/dm9000.c）、プラットフォームデバイスで、ターゲットプラットフォーム初期化コードに、該当の構造を書き込む、具体的な手順は下記の通り

まずドライバに必要のヘッダファイル dm9000.h を追加する

```
#include <linux/dm9000.h>
```

DM9000 の LAN カードデバイスデバイスの物理ベースアドレスを定義する。

```
/* DM9000AEP 10/100 ethernet controller */
#define MACH_MINI2440_DM9K_BASE (S3C2410_CS4 + 0x300)
```

次に DM9000 NIC ドライバインタフェースと合わせ、該当プラットフォームのデバイスリソースを修正し、下記のとおりである

```
static struct resource mini2440_dm9k_resource[] = {
        [0] = {
                .start = MACH_MINI2440_DM9K_BASE,
                .end = MACH_MINI2440_DM9K_BASE + 3,
                .flags = IORESOURCE_MEM
        },
```

```
        [1] = {
                .start = MACH_MINI2440_DM9K_BASE + 4,
                .end = MACH_MINI2440_DM9K_BASE + 7,
                .flags = IORESOURCE_MEM
        },
        [2] = {
                .start = IRQ_EINT7,
                .end = IRQ_EINT7,
                .flags = IORESOURCE_IRQ | IORESOURCE_IRQ_HIGHEDGE,
        }
};
/*
* * * The DM9000 has no eeprom, and it's MAC address is set by
* * * the bootloader before starting the kernel.
* * */
static struct dm9000_plat_data mini2440_dm9k_pdata = {
        .flags =                (DM9000_PLATF_16BITONLY | DM9000_PLATF_NO_EEPROM),
};

static struct platform_device mini2440_device_eth = {
        .name                = "dm9000",
        .id                  = -1,
        .num_resources       = ARRAY_SIZE(mini2440_dm9k_resource),
        .resource            = mini2440_dm9k_resource,
        .dev =               {
                .platform_data = &mini2440_dm9k_pdata,
        },
};
```

mini2440 に NIC デバイスを加えます：

```
static struct platform_device *mini2440_devices[] __initdata = {
        &s3c_device_usb,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c0,
        &s3c_device_iis,
        &mini2440_device_eth,
        &s3c_device_nand,
};
```

このように DM9000 プラットフォームデバイスのインタフェースは修正された

3.9.2 DM9000 が使用するビット幅レジスタ調整

Linux-2.6.32.2 の DM9000 NIC ドライバは mini2440 専用ではないため、ソースコード移植が必要である：

linux-2.6.32.2/drivers/net/dm9000.c を開き、ヘッダファイルに 2410 の関連設定定義を追加し、次の赤い部分のように示す

```
#include <asm/delay.h>
#include <asm/irq.h>
#include <asm/io.h>

#if defined(CONFIG_ARCH_S3C2410)
#include <mach/regs-mem.h>
#endif

#include "dm9000.h"
```

dm9000 デバイスの初期化関数には次の赤い部分を追加し、これは DM9000 が使用するチップセレクトバスを設定するタイミングで、現在、mini2440 にはバスを通じて外部へ拡張するデバイスは一つだけで、このデバイスで関連のレジスタ設定を直接に修正できる。mach-mini2440.c でも修正できる。

```
static int __init
dm9000_init(void)
{
#if defined(CONFIG_ARCH_S3C2410)
        unsigned int oldval_bwscon = *(volatile unsigned int *)S3C2410_BWSCON;
        unsigned int oldval_bankcon4 = *(volatile unsigned int *)S3C2410_BANKCON4;
        *((volatile unsigned int *)S3C2410_BWSCON) =
                        (oldval_bwscon & ~(3<<16)) | S3C2410_BWSCON_DW4_16 |
S3C2410_BWSCON_WS4 | S3C2410_BWSCON_ST4;
        *((volatile unsigned int *)S3C2410_BANKCON4) = 0x1f7c;
#endif
        printk(KERN_INFO "%s Ethernet Driver, V%s\n", CARDNAME, DRV_VERSION);

        return platform_driver_register(&dm9000_driver);
}
```

3.9.3 MAC アドレスについて

本開発ボードの使用する DM9000 LAN カードは EEPROM と外部接続し MAC アドレスを保存しないから、システムの MAC アドレスは"ソフト"アドレス（広範囲）であり、即ちソフトウェアを通じて、任意値を修正できる、static int __devinit dm9000_probe(struct platform_device *pdev)関数から見て：

```
/* try reading the node address from the attached EEPROM */
; EEPROM から MAC アドレスを読み取り
for (i = 0; i < 6; i += 2)
        dm9000_read_eeprom(db, i / 2, ndev->dev_addr+i);

if (!is_valid_ether_addr(ndev->dev_addr) && pdata != NULL) {
        mac_src = "platform data";
        memcpy(ndev->dev_addr, pdata->dev_addr, 6);
}

if (!is_valid_ether_addr(ndev->dev_addr)) {
        /* try reading from mac */

         mac_src = "chip";
         for (i = 0; i < 6; i++)
                 ndev->dev_addr[i] = ior(db, i+DM9000_PAR);
}
; “ソフト” MAC アドレス: 08:90:90:90:90:90
memcpy(ndev->dev_addr, "¥x08¥x90¥x90¥x90¥x90¥x90", 6);

if (!is_valid_ether_addr(ndev->dev_addr))
         dev_warn(db->dev, "%s: Invalid ethernet MAC address. Please "
                 "set using ifconfig¥n", ndev->name);
```

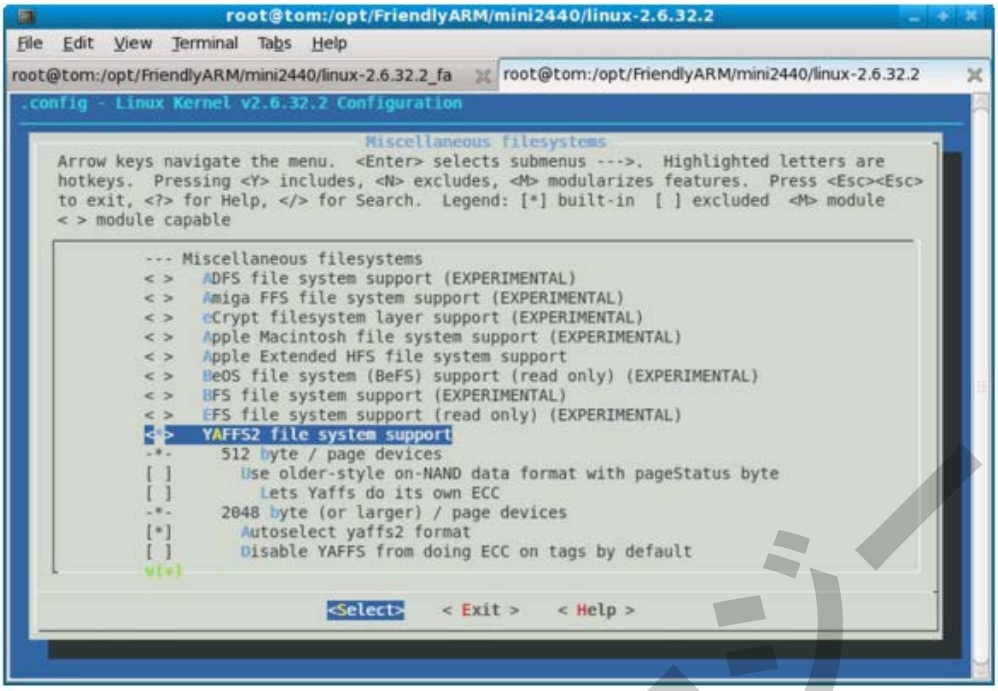
ここまで DM9000 の移植は完了する

3.9.4　カーネル設定、DM9000 を追加、コンパイルとテスト

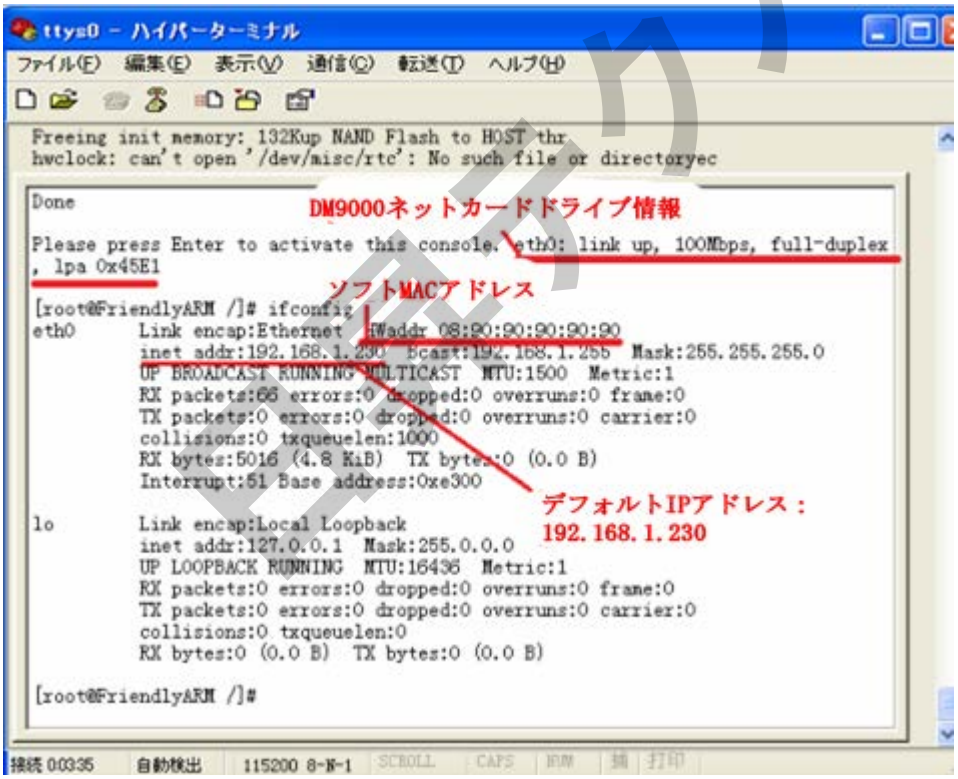
カーネルソースディレクトリで、#make　menuconfig を実行し、カーネルに NIC ドライバを設定し始め、下記のメニューオプションを順に選択する

```
Device Drivers  --->Network device support  ---> Ethernet (10 or 100Mbit)  --->
```

DM9000 の設定オプションを見つけ、DM9000 は選択されたと見えるのは、Linux-2.6.32.2 のデフォルトカーネル設定は DM9000 のサポートを追加してからになる



次に#make zImage を実行し、最後に arch/arm/boot/zImage ファイルを生成し、｀k｀コマンドを使用し、開発ボードにプログラミングし、デフォルトファイルシステムで起動し、コマンドラインターミナルに ifconfig コマンドを実行する。下記図を参照することを見える



3.10 RTC ドライバを活性化

3.10.1 初期化ファイルに RTC デバイス構造を追加

Linux-2.6.32.2 カーネルは 2440 の RTC ドライバに対して十分であるが、mach-mini2440.c のデバイスリストに追加しないため、活性化されていない。RTC 構造を追加し、次の赤い内容のように示す

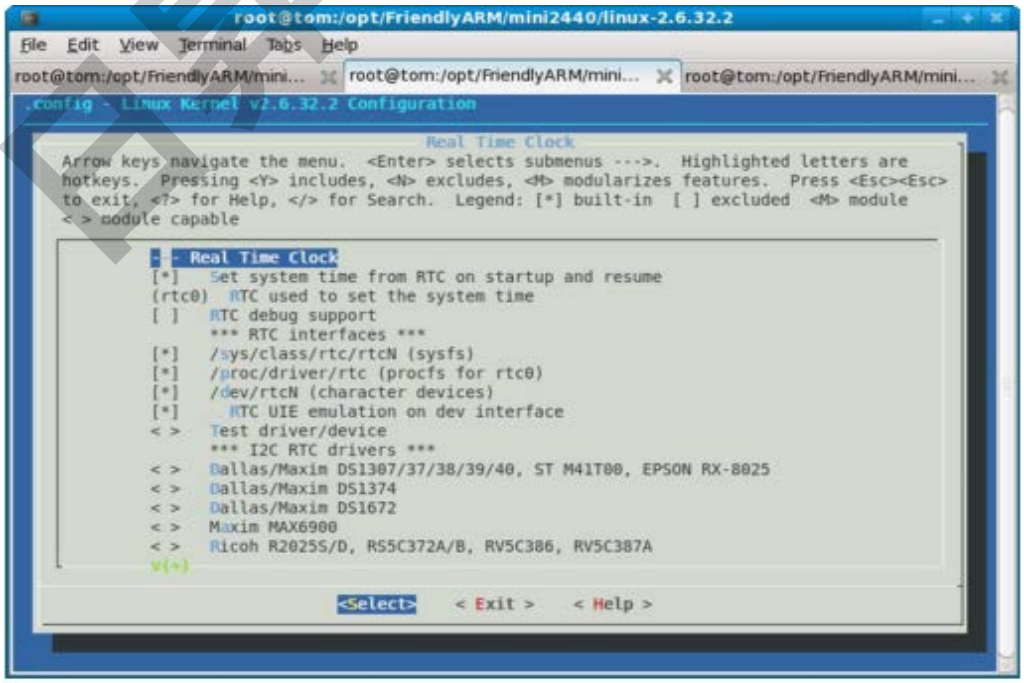
```
;mini2440 に RTC デバイス仕組みを追加
static struct platform_device *mini2440_devices[] __initdata = {
        &s3c_device_usb,
        &s3c_device_rtc,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c0,
        &s3c_device_iis,
        &mini2440_device_eth,
        &s3c_device_nand,
};
```

3.10.2 カーネルに RTC を設定

カーネルを再構成、RTC のドライバサポートを追加する、下記のメニューオプションを順に選択する

```
Device Drivers  --->
  <*> Real Time Clock  --->
```

図のようなメニューが出る



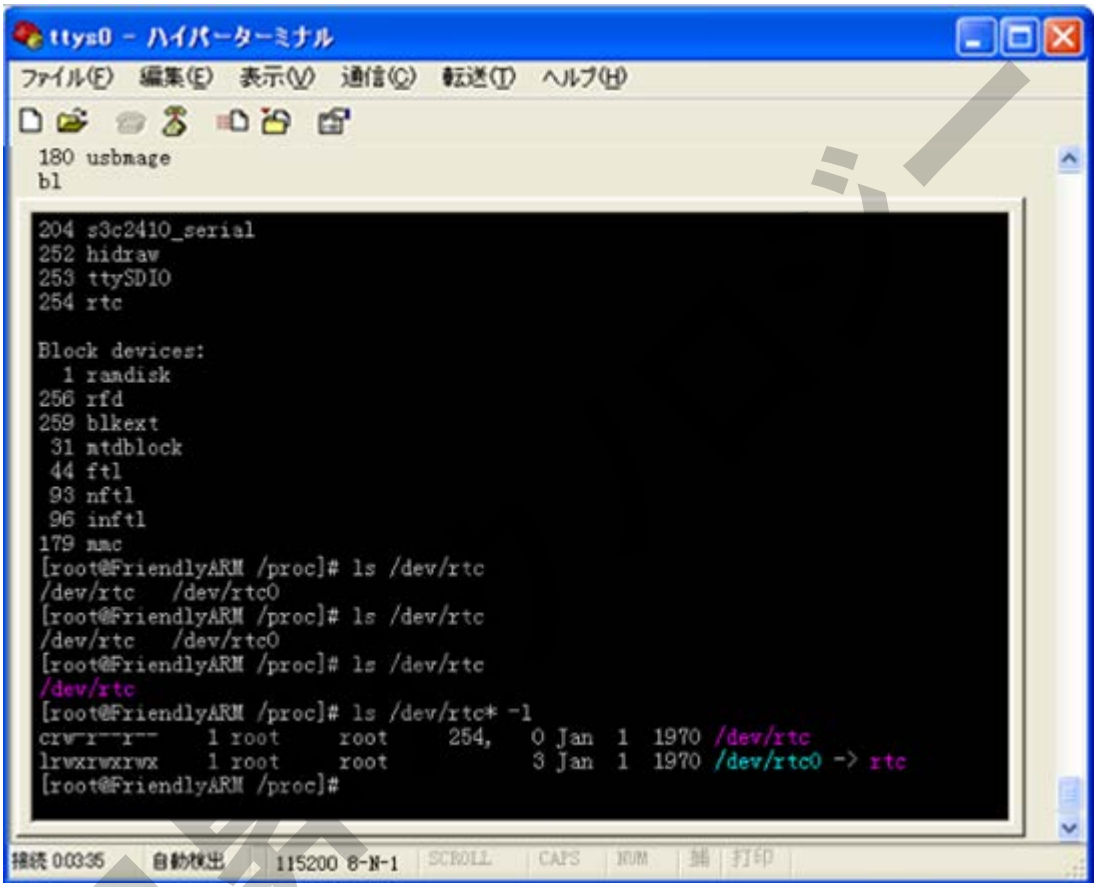
デフォルト設定は RTC の関連オプションを設定メニューの一番に選択された

〈*〉 Samsung S3C series SoC RTC オプション

カーネルの 2440 の RTC ドライバ設定オプションである。

3.10.3 RTC をテスト

カーネル設定メニューを終了し、#make zImage を実行し、生成した arch/arm/boot/zImage を開発ボードにプログラミングし、/dev ディレクトリ下に/dev/rtc デバイスドライバが確認できる、下図を参照する



RTC をテストしようとすれば、mini2440 ユーザーマニュアルの関連章節を参照する：

Linux では、普通は date コマンドで時間を変更する、S3C2440 内部クロックを linux システムクロックと同期させるため、hwclock コマンドを使用する：

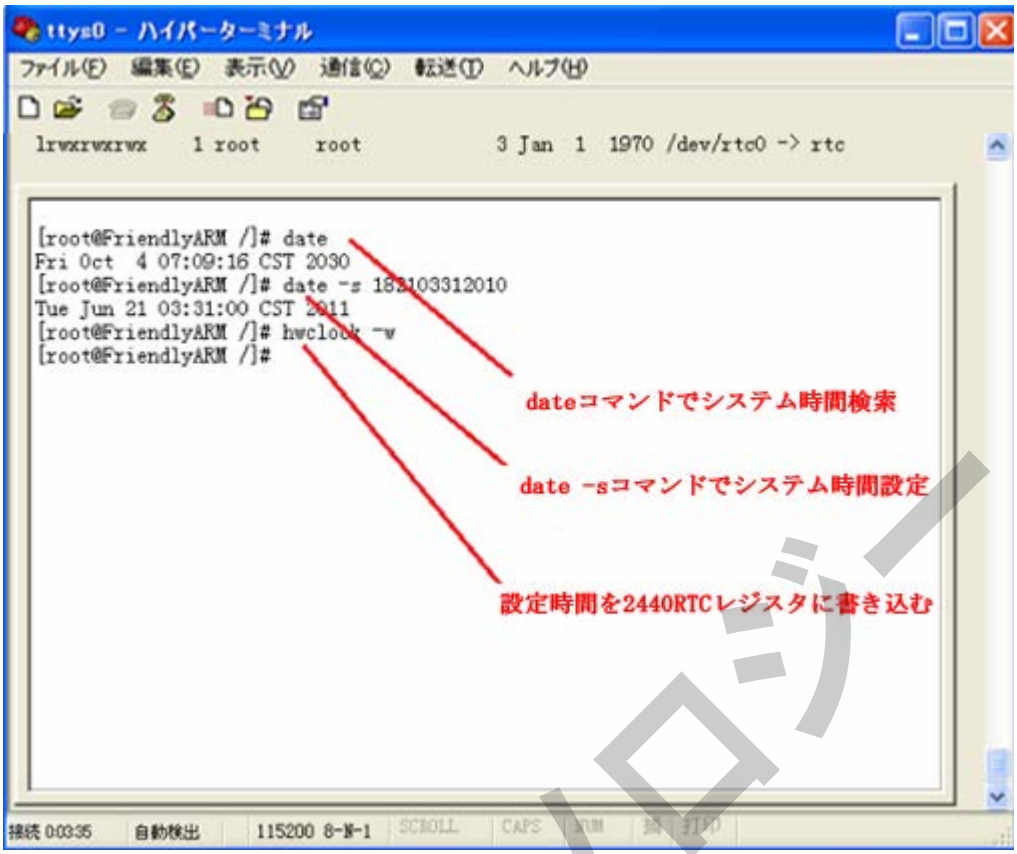
(1) date -s 042916352007 #時間を 2007-04-29 16:34 と設定する

(2) hwclock -w #先設定した時間を S3C2440 内部の RTC に保存する

(3) オンモードにする時、hwclock -s コマンドで linux システムクロックを RTC に回復できる、基本的には該当コマンドは/etc/init.d/rcS ファイルに置かれ、自動的に実行させる

注：既存システムは既に hwclock -s コマンドを rcS ファイルに書き込まれました。

実行図は下記の通り

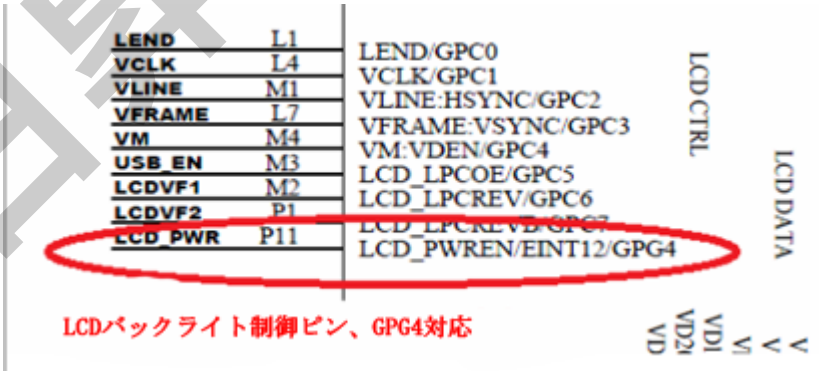


3.11 LCD バックライト･ドライバを追加

3.11.1 LCD バックライト･制御原理

今までコマンドラインで移植作業を行い、結果を表示される、LCD スクリーンには何の表示もない。本節からこれについて説明する

mini2440/micro2440 開発ボードには LCD バックライトは CPU の LCD_PWR ピンを通じて制御し、GPG4 と対応する。下図を参照する



LCD_PWR 出力はハイレベル 奇数｀1｀になる場合、バックライトをオンし、出力はローレベル 偶数｀0｀になる場合、バックライトをオフする（注：ここで、バックライトをオン/オフ操作するだけ、バックライト輝度を調整しない）

3.11.2 カーネルにバックライトドライバを追加

現在、ソフトウェアでバックライトの制御スイッチを制御するため、バックライトドライバを追加する。

実現したい効果:

コマンドターミナルにバックライトデバイスへ偶数`0`を送信するすると、バックライトをオフできる、奇数`1`を送信するすると、バックライトをオンできる。ユーザーマニュアル（LCD バックライトを制御）:

注：LCD バックライトデバイスファイル：/dev/backlight

コマンドライン：echo 0 > /dev/backlight を入力する、バックライトをオフできる

コマンドライン：echo 1 > /dev/backlight を入力する、バックライトをオンできる

そして、linux-2.6.32.2/drivers/videoディレクトリ下にmini2440_backlight.cファイルを追加する、内容は下記の通り

```
;下記のヘッドファイルは全部必要ではありません、余ったコードはプログラムに影響はありません
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/input.h>
#include <linux/init.h>
#include <linux/serio.h>
#include <linux/delay.h>
#include <linux/clk.h>
#include <linux/miscdevice.h>
#include <linux/gpio.h>

#include <asm/io.h>
#include <asm/irq.h>
#include <asm/uaccess.h>
#include <mach/regs-clock.h>
#include <plat/regs-timer.h>

#include <mach/regs-gpio.h>
#include <linux/cdev.h>

#undef DEBUG
//#define DEBUG
#ifdef DEBUG
```

```
#define DPRINTK(x...) {printk(__FUNCTION__"(%d): ",__LINE__);printk(##x);}
#else
#define DPRINTK(x...) (void)(0)
#endif

;バックライトドライバを backligh と定義し、/dev/backlight に保存します
#define DEVICE_NAME
"backlight"

;バックライト変数を bl_state と定義し、バックライトのオン/オフ状態を記録します
static unsigned int bl_state;

;バックライトスイッチ関数を設定し、バックライト変数を逆転します
static inline void set_bl(int state)
{
        bl_state = !!state; // bl_state 変数逆転
        s3c2410_gpio_setpin(S3C2410_GPG(4), bl_state); //レジスタ GPG4 に結果を書き込む
}

;バックライトステータス読み取り
static inline unsigned int get_bl(void)
{
        return bl_state;
}

; アプリからパラメータを読み取り、カーネルに伝送します
static ssize_t dev_write(struct file *file, const char *buffer, size_t count, loff_t * ppos)
{
        unsigned char ch;
        int ret;
        if (count == 0) {
                return count;
        }
        ; copy_from_user 関数でユーザーレイヤ/アプリレイヤからパラメータを読み取り
        ret = copy_from_user(&ch, buffer, sizeof ch) ? -EFAULT : 0;
        if (ret) {
                return ret;
        }

        ch &= 0x01; //奇数か偶数かを判断
```

```
        set_bl(ch); //バックライトステータス設定

        return count;
}

;カーネルパラメータをユーザーレイヤ/アブリレイヤの read 関数に伝送します
static ssize_t dev_read(struct file *filp, char *buffer, size_t count, loff_t *ppos)
{
        int ret;
        unsigned char str[] = {'0', '1' };

        if (count == 0) {
                return 0;
        }

        ; copy_to_user 関数でカーネルパラメータをユーザーレイヤ/アブリレイヤに伝送します
        ret = copy_to_user(buffer, str + get_bl(), sizeof(unsigned char) ) ? -EFAULT : 0;
        if (ret) {
                return ret;
        }

        return sizeof(unsigned char);
}

; デバイス操作セット
static struct file_operations dev_fops = {
        owner:    THIS_MODULE,
        read:dev_read,
        write:    dev_write,
};

static struct miscdevice misc = {
        .minor = MISC_DYNAMIC_MINOR,
        .name = DEVICE_NAME,
        .fops = &dev_fops,
};

; デバイス初期化、ブートカーネルを有効にします
static int __init dev_init(void)
```

```
{
        int ret;

        ret = misc_register(&misc);

        printk (DEVICE_NAME"¥tinitialized¥n");

        ;バックライトピン初期化、 GPG4 を出力
        s3c2410_gpio_cfgpin(S3C2410_GPG(4), S3C2410_GPIO_OUTPUT);
        ; ブートカーネル時、バックライト・オン
        set_bl(1);
        return ret;
}

static void __exit dev_exit(void)
{
        misc_deregister(&misc);
}

module_init(dev_init); //バックライト･ドライバ･モジュール登録
module_exit(dev_exit); //バックライト･ドライバ･モジュールアンインストール
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
```

次にバックライト設定オプションをカーネル設定メニューに追加し、
linux-2.6.32.2/drivers/video/Kconfig をオープンし、下図を参照し位置に追加する

```
config FB_S3C2410_DEBUG
        bool "S3C2410 lcd debug messages"
        depends on FB_S3C2410
        help
            Turn on debugging messages. Note that you can set/unset at run time
            through sysfs

#MINI2440 に バックライト・ドライバ・コンフィギュレーションを加えます
config BACKLIGHT_MINI2440
        tristate "Backlight support for mini2440 from FriendlyARM"
        depends on MACH_MINI2440 && FB_S3C2410
        help
            backlight driver for MINI2440 from FriendlyARM
```

```
config FB_SM501
        tristate "Silicon Motion SM501 framebuffer support"
        depends on FB && MFD_SM501
        select FB_CFB_FILLRECT
        select FB_CFB_COPYAREA
        select FB_CFB_IMAGEBLIT
```

linux-2.6.32.2/drivers/video/Makefile をオープンし、設定定義に基づき、ドライバターゲットファイルを追加し、下図を参照する

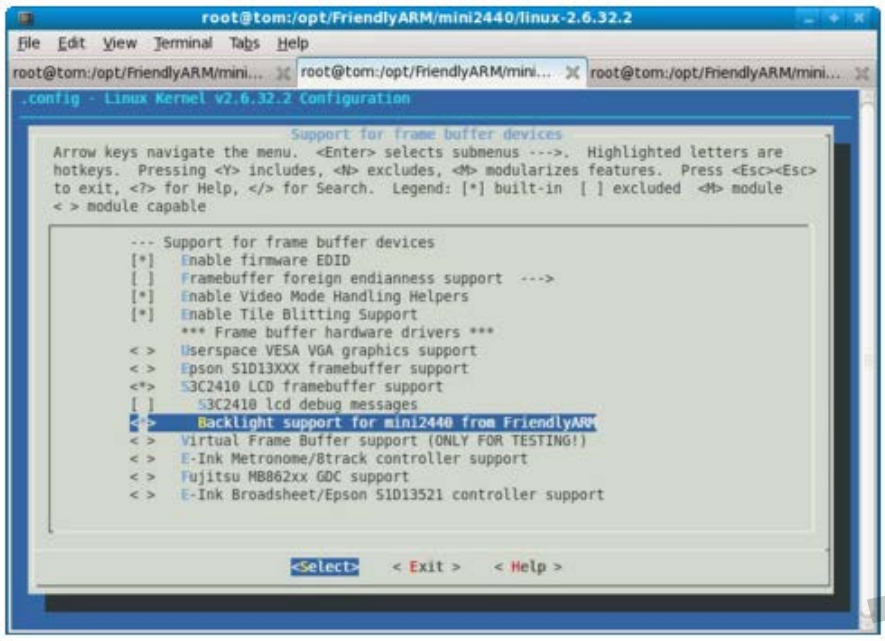
```
# the test framebuffer is last
obj-$(CONFIG_FB_VIRTUAL)                += vfb.o

#video output switch sysfs driver
obj-$(CONFIG_VIDEO_OUTPUT_CONTROL) += output.o

obj-$(CONFIG_BACKLIGHT_MINI2440) += mini2440_backlight.o
```

ここまでカーネルに mini2440 のバックライトドライバは移植された、カーネルソースコードルートディレクトリに make menuconfig を実行し、次のサブメニューを順に選択する

```
Device Drivers --->
 Graphics support --->
      <*> Support for frame buffer devices --->
```

該当設定オプションを探し、下図を参照する

カーネルソースコードルートディレクトリに make menuconfig を実行し、次のサブメニューを順に選択する。該当設定オプションを探し、下図を参照する



スペースを押し、先に追加された mini2440 オプションを選択、保存、カーネル設定メニューを終了し、コマンドライン：make zImage を実行し、arch/arm/boot/zImage を生成する、supervivi の゛k゛機能を使用し、開発ボードにプログラミング、起動すると、バックライトはオンにする。

3.12 LCD ディスプレイドライバを移植

3.12.1 LCD ドライバ基礎知識

Linux-2.6.32.2 カーネルは S3C2440 の LCD コントローラドライバをサポートした。先ず、2440 LCD コントローラとドライバ関連の LCD 基礎知識を説明する

注：TFT LCD、即ちトゥルーカラーを紹介する。

LCD ドライバの最も肝心な点はクロック周波数(Clock frequency)の設定のことで、クロック周波数の設定は間違えたら LCD のディスプレイは点滅、または表示しない場合がある。一方、LCD の Datasheet にはデフォルト周波数があり、例えば mini2440 に使用する Toppoly 3.5゛LCD、データマニュアルは下記の通り：

Display Mode	Parameter	Symbol	Conditions	Ratings			Unit
				MIN	TYP	MAX	
Normal	Vertical cycle	VP		323	326	340	Line
	Vertical data start	VDS	VS+VBP	2	4	—	Line
	Vertical front porch	VFP		1	2	—	Line
	Vertical blanking period	VBL	VS+VBP+VFP	3	6	—	Line
	Vertical active area	VDISP		—	320	—	Line
	Horizontal cycle	HP		260	280	300	dot
	Horizontal front porch	HFP		4	10	—	dot
	Horizontal Sync Pulse width	HS		8	10	—	dot
	Horizontal Back porch	HBP		18	20	—	dot
	Horizontal Data start	HDS	HS+HBP	26	30	—	dot
	Horizontal active area	HDISP		240	240	240	dot
	Clock frequency	fclk		5.02	6.39	6.85	MHz
		tclk		199	156	146	nS

デフォルト周波数は 6.39MHz で、6.4MHz に近く、範囲は 5M-6.85MHz で、S3C2440 の LCD コントローラと

関連設定は CLKVAL で、設定を通じて、LCD インタフェースの VCLK ピンに LCD の必要なクロック周波数を生成する。CLKVAL と VCLK の関係は 2440 データシートで（411 ページ）、次の説明がある

> The rate of VCLK signal depends on the CLKVAL field in the LCDCON1 register. Table 15-3 defines the relationship of VCLK and CLKVAL. The minimum value of CLKVAL is 0

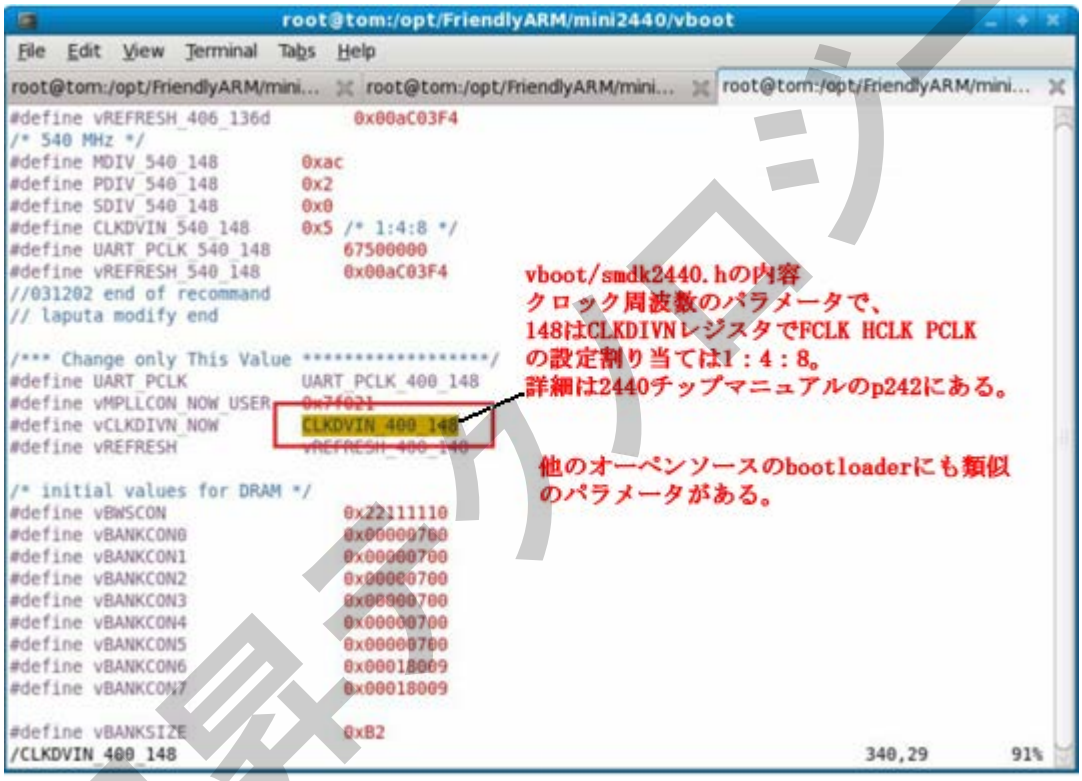
関連数学関係式は下記の通り

$$VCLK(Hz) = HCLK/[(CLKVAL+1)x2]$$

結論

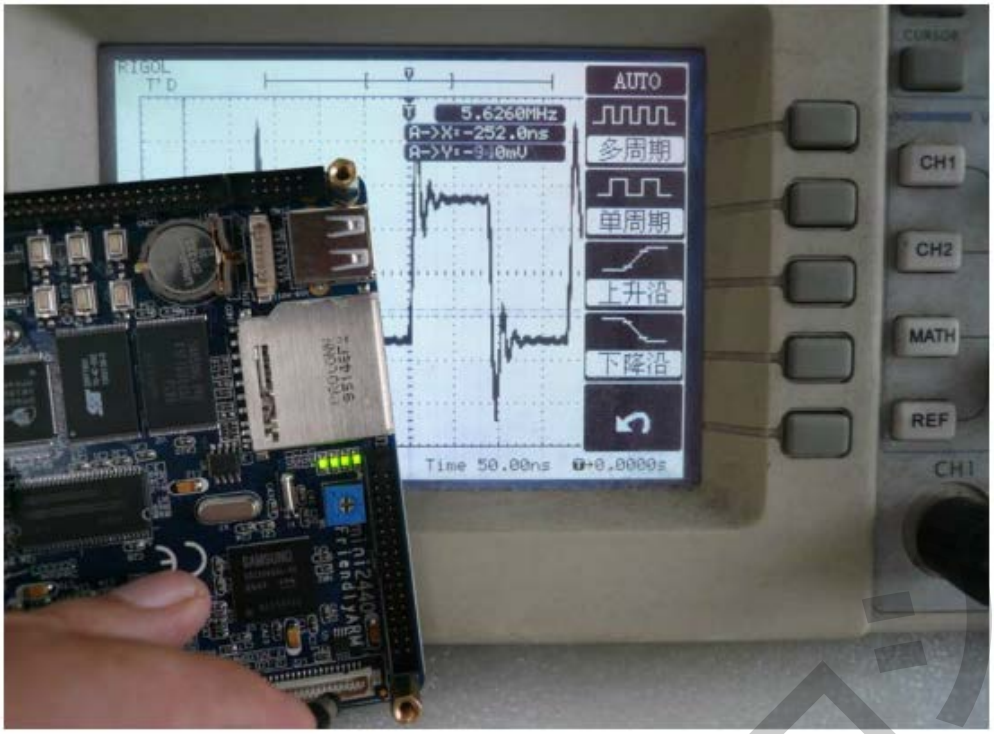
$$VCLK = HCLK / ((CLKVAL+1)*2)$$

HCLK について、開発ボードは 400Mhz で実行し、bootloader のソースコードのヘッダファイルで分かる、下図を参照する



上記により：FCLK:HCLK:PCLK = 1:4:8、結論 HCLK=100Mhz、上述の公式より CLKVAL は CLKVAL=HCLK/(VCLK*2)-1、CLKVAL = 100000000 / (6400000 * 2) - 1 = 6.8 の結果が出る。

一番近い整数値 7 を選択し、LCDCON1:17-8 に書き込み（注：実際値は 8 である）、これによって生成された VCLK 周波数、実際テストは 5.63Mhz ぐらいで、これは 5-6.85Mhz の間の値であり、下図を参照する



3.12.2　新しいカーネル pixclock パラメータ

旧バージョンの Linux カーネルの LCD レジスタの設定は直接 CLKVAL をコンパイルするが、Linux-2.6.32.2 カーネルからは゛pixclock゛というパラメータで調整する。これによって、計算が複雑となり、実際操作では誤差があると思われる。

注：本章提供した pixclock パラメータは CLKVAL の値により繰り返しのテストで得るもので、推論計算の値ではありません。

Framebuffer ドライバ（linux-2.6.32.2/ drivers/video/s3c2410fb.c)には次の関数がある

clkdiv = DIV_ROUND_UP(s3c2410fb_calc_pixclk(fbi、 var->pixclock)、 2);

ここの clkdiv は上記の CLKVAL となり、DIV_ROUND_UP は一つのマクロ定義で、include/linux/kernel.h ファイルにある。

#define DIV_ROUND_UP(n、d) (((n) + (d) - 1) / (d))

実はこれは数学的な概念であり：整数を切り上げ。次は゛上へ整数を切り上げ゛についての説明である

本アルゴリズムに対して、除数は〝2〟で、〝(n/2)+0.5〟を対応するする整数値と理解できる。だから、ここで、誤差を予想される。即ち、n 値は一定の範囲があり、ここの n は〝s3c2410fb_calc_pixclk(fbi、var->pixclock)〟であり、だから上記の公式は次のように変更されできる clkdiv= s3c2410fb_calc_pixclk(fbi、 var->pixclock)/2 + 0.5、そして s3c2410fb_calc_pixclk(fbi、

1. 問題

A,B は整数であり、そして A>1, B>1

┌A / B┐すなわち A / B の値を上丸めます。

A/B が割り切れる時は、最終値は A/B の値です。

割り切れない場合、戻り値は int(A/B) + 1

算式の使用例：ダイナミックに増やすバッファが一つであり、増やすステップは B,増やすバッファのサイズは A にしたい場合、本算数で A のサイズを計算し、増やすバッファのサイズを適合に割り当てられる。余った部分は B より多くない 2. 方法

int((A+B-1)/B)

3. HUNTON の証明

上丸めは UP で表示します

A>1、B>1；そして A、B は共に整数であり、A=NB+M

N は負でない整数、M は 0 から B-1 までの数字、

A/B = N + M/B

(A+B-1)/B = N + 1 + (M - 1)/B;

M は 0 になる時、

UP(A/B) = N、

int((A+B-1)/B) = N + int(1 - 1/B) = N

M は 1 ～ B-1 の範囲の場合、0 <= M-1 <= B-2

UP(A/B) = N + 1、

int((A+B-1)/B) = N + 1 + int((M-1)/B) = N + 1

即ち A>1、B>1 の整数に A、B ：

var->pixclock)関数は inux-2.6.32.2/drivers/video/s3c2410fb.c に次のように定義される

```
/* s3c2410fb_calc_pixclk()
*
* calculate divisor for clk->pixclk
*/
static unsigned int s3c2410fb_calc_pixclk(struct s3c2410fb_info *fbi,
                                          unsigned long pixclk)
{
        unsigned long clk = fbi->clk_rate;
        unsigned long long div;

        /* pixclk is in picoseconds, our clock is in Hz
         *
        * Hz -> picoseconds is / 10^-12
        */
       ;関数計算の結果
       div = (unsigned long long)clk * pixclk;
       div >>= 12; /* div / 2^12 */
       do_div(div, 625 * 625UL * 625); /* div / 5^12 */

       dprintk("pixclk %ld, divisor is %ld¥n", pixclk, (long)div);
       return div;
}
```

clkdiv=clk*pixclk/(10^12)/2+0.5 を出し、プリントアウトした結果を検証したら、ここの clk は HCLK である、static void s3c2410fb_activate_var(struct fb_info *info)関数の説明より次の関係を出し：

CLKVAL=clkdiv-1

2440 データシートから得る公式 CLKVAL=HCLK/(VCLK*2)-1 と結び付け、次のﾞソフトﾞ結果が出る（ﾞソフトﾞは誤差範囲）

Pixclk=(HCLK-VLCK)x10^12/HCLK*VCLK

Toppoly 画面を例として

HCLK=100Mhz=100、000、000Hz

VLCK=6.4Mhz=6400、000Hz

計算は：pixclk =146250、単位は ps(picoseconds)で、実際の設定値 170000 と誤差がある

その他 Linux カーネルドキュメントには既に一つ pixclock を計算する方法があり、linux/Documentation/fb/framebuffer.txt を参照する。

パラメータに対して理解し難い方々は、検証済みの移植パラメータで設定する方法をお勧める。

3.12.3　カーネルに各種 LCD タイプのサポートを追加

arch/arm/mach-s3c2440/mach-mini2440.c をオープンし、最初の LCD デバイスプラットフォームのコードを削除する

```
/* LCD driver info */

static struct s3c2410fb_display smdk2440_lcd_cfg __initdata = {

.       lcdcon5                 = S3C2410_LCDCON5_FRM565
```

```
                S3C2410_LCDCON5_INVVLINE |
                S3C2410_LCDCON5_INVVFRAME |
                S3C2410_LCDCON5_PWREN |
                S3C2410_LCDCON5_HWSWP,

        .type           = S3C2410_LCDCON1_TFT,

        .width          = 240,
        .heigh          t = 320,

        .pixclock       = 166667, /* HCLK 60 MHz, divisor 10 */
        .xres           = 240,
        .yres           = 320,
        .bpp            = 16,
        .left_margin    = 20,
        .right_margin = 8,
        .hsync_len      = 4,
        .upper_margin = 8,
        .lower_margin = 7,
        .vsync_len      = 4,
};
static struct s3c2410fb_mach_info smdk2440_fb_info __initdata = {
        .displays = &smdk2440_lcd_cfg,
        .num_displays = 1,
        .default_display = 0,

#if 0
        /* currently setup by downloader */
        .gpccon         = 0xaa940659,
        .gpccon_mask = 0xffffffff,
        .gpcup          = 0x0000ffff,
        .gpcup_mask = 0xffffffff,
        .gpdcon         = 0xaa84aaa0,
        .gpdcon_mask = 0xffffffff,
        .gpdup          = 0x0000faff,
        .gpdup_mask = 0xffffffff,
#endif
        .lpcsel         = ((0xCE6) & ~7) | 1<<4,
};
```

移植コードを追加する、次のように示す

```
/* LCD driver info */

;NEC 3.5”LCD コンフィギュレーション/パラメータ
#if defined(CONFIG_FB_S3C2410_N240320)

#define LCD_WIDTH 240
#define LCD_HEIGHT 320
#define LCD_PIXCLOCK 100000

#define LCD_RIGHT_MARGIN 36
#define LCD_LEFT_MARGIN 19
#define LCD_HSYNC_LEN 5

#define LCD_UPPER_MARGIN 1
#define LCD_LOWER_MARGIN 5
#define LCD_VSYNC_LEN 1

;シャップ 8”LCD コンフィギュレーション/パラメータ
#elif defined(CONFIG_FB_S3C2410_TFT640480)
#define LCD_WIDTH 640
#define LCD_HEIGHT 480
#define LCD_PIXCLOCK 80000

#define LCD_RIGHT_MARGIN 67
#define LCD_LEFT_MARGIN 40
#define LCD_HSYNC_LEN 31

#define LCD_UPPER_MARGIN 25
#define LCD_LOWER_MARGIN 5
#define LCD_VSYNC_LEN 1

; Toppoly 3.5”LCD コンフィギュレーション/パラメータ
#elif defined(CONFIG_FB_S3C2410_T240320)
#define LCD_WIDTH 240
#define LCD_HEIGHT 320
#define LCD_PIXCLOCK 146250//170000
```

```
#define LCD_RIGHT_MARGIN 25
#define LCD_LEFT_MARGIN 0
#define LCD_HSYNC_LEN 4

#define LCD_UPPER_MARGIN 1
#define LCD_LOWER_MARGIN 4
#define LCD_VSYNC_LEN 1

; Innolux 7"LCD コンフィギュレーション/パラメータ
#elif defined(CONFIG_FB_S3C2410_TFT800480)
#define LCD_WIDTH 800
#define LCD_HEIGHT 480
#define LCD_PIXCLOCK 11463//40000

#define LCD_RIGHT_MARGIN 67
#define LCD_LEFT_MARGIN 40
#define LCD_HSYNC_LEN 31

#define LCD_UPPER_MARGIN 25
#define LCD_LOWER_MARGIN 5
#define LCD_VSYNC_LEN 1

;LCD2VGA(解像度 1024x768)モジュールコンフィギュレーション/パラメータ
#elif defined(CONFIG_FB_S3C2410_VGA1024768)
#define LCD_WIDTH 1024
#define LCD_HEIGHT 768
#define LCD_PIXCLOCK 80000

#define LCD_RIGHT_MARGIN 15
#define LCD_LEFT_MARGIN 199
#define LCD_HSYNC_LEN 15

#define LCD_UPPER_MARGIN 1
#define LCD_LOWER_MARGIN 1
#define LCD_VSYNC_LEN 1
#define LCD_CON5 (S3C2410_LCDCON5_FRM565 | S3C2410_LCDCON5_HWSWP)

#endif
```

```
#if defined (LCD_WIDTH)

static struct s3c2410fb_display mini2440_lcd_cfg __initdata = {

#if !defined (LCD_CON5)
        .lcdcon5        = S3C2410_LCDCON5_FRM565 |
                          S3C2410_LCDCON5_INVVLINE |
                          S3C2410_LCDCON5_INVVFRAME |
                          S3C2410_LCDCON5_PWREN |
                          S3C2410_LCDCON5_HWSWP,
#else
        .lcdcon5        = LCD_CON5,
#endif
        .type           = S3C2410_LCDCON1_TFT,

        .width          = LCD_WIDTH,
        .height         = LCD_HEIGHT,

        .pixclock       = LCD_PIXCLOCK,
        .xres           = LCD_WIDTH,
        .yres           = LCD_HEIGHT,
        .bpp            = 16,
        .left_margin    = LCD_LEFT_MARGIN + 1,
        .right_margin   = LCD_RIGHT_MARGIN + 1,
        .hsync_len      = LCD_HSYNC_LEN + 1,
        .upper_margin   = LCD_UPPER_MARGIN + 1,
        .lower_margin   = LCD_LOWER_MARGIN + 1,
        .vsync_len      = LCD_VSYNC_LEN + 1,
};

static struct s3c2410fb_mach_info mini2440_fb_info __initdata = {
        .displays       = &mini2440_lcd_cfg,
        .num_displays   = 1,
        .default_display  = 0,

        .gpccon         = 0xaa955699,
        .gpccon_mask    = 0xffc003cc,
        .gpcup          = 0x0000ffff,
```

```
            .gpcup_mask = 0xffffffff,

            .gpdcon = 0xaa95aaa1,
            .gpdcon_mask = 0xffc0fff0,
            .gpdup = 0x0000faff,
            .gpdup_mask = 0xffffffff,

            .lpcsel = 0xf82,
};

#endif
```

次に drivers/video/Kconfig をオープンし、1935 行ぐらいの位置でに次の設定情報を追加する

```
config FB_S3C2410_DEBUG
    bool "S3C2410 lcd debug messages"
    depends on FB_S3C2410
    help
        Turn on debugging messages. Note that you can set/unset at run time
        through sysfs

choice
    prompt "LCD select"
    depends on FB_S3C2410
    help
        S3C24x0 LCD size select

config FB_S3C2410_T240320
    boolean "3.5 inch 240X320 Toppoly LCD"
    depends on FB_S3C2410
    help
        3.5 inch 240X320 Toppoly LCD

config FB_S3C2410_N240320
    boolean "3.5 inch 240X320 NEC LCD"
    depends on FB_S3C2410
    help
        3.5 inch 240x320 NEC LCD

config FB_S3C2410_TFT640480
```

```
    boolean "8 inch 640X480 L80 LCD"
    depends on FB_S3C2410
    help
        8 inch 640X480 LCD

config FB_S3C2410_TFT800480
    boolean "7 inch 800x480 TFT LCD"
    depends on FB_S3C2410
    help
        7 inch 800x480 TFT LCD

config FB_S3C2410_VGA1024768
    boolean "VGA 1024x768"
    depends on FB_S3C2410
    help
        VGA 1024x768

Endchoice

config BACKLIGHT_MINI2440
    tristate "Backlight support for mini2440 from FriendlyARM"
    depends on MACH_MINI2440 && FB_S3C2410
    help
        backlight driver for MINI2440 from FriendlyARM

config FB_SM501
    tristate "Silicon Motion SM501 framebuffer support"
    depends on FB && MFD_SM501
    select FB_CFB_FILLRECT
    select FB_CFB_COPYAREA
    select FB_CFB_IMAGEBLIT
```


これでLCDドライバの移植が完了する。上述方式で他型のLCDドライバも追加できる。小サイズのpixclockパラメータはToppoly 3.5 ″を参照し、640x480解像度以上のタイプのパラメータは8` LCDのを参照し、使用LCDの長さ・幅も変更する必要がある。

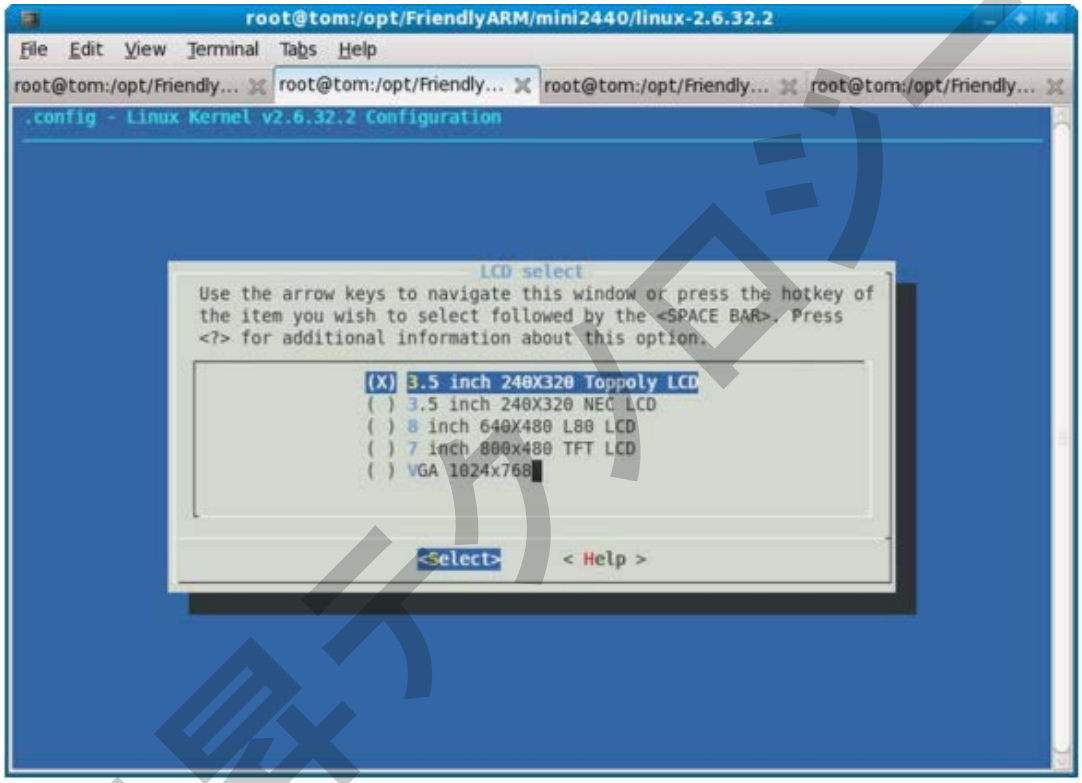
3.12.4　カーネルを設定し、開発ボードにダウンロードし、テストする

コマンドライン：make menuconfigを入力し、カーネル設定に入り、次のサブメニューに従い順に選択す

```
Device Drivers --->
 Graphics support --->
    <*> Support for frame buffer devices --->
        LCD select (3.5 inch 240X320 Toppoly LCD) --->
```

る

下図を参照する LCD モデルの設定オプションが出る



space または Enter ボタンを押して必要の LCD モデルを選択し、カーネル設定を保存し、終了する

コマンドライン：#make zImage を実行し、arch/arm/boot/zImage を生成し、開発ボードにプログラミングし、スクリーンには小さなペンギンがでると見えます。

3.13 Linux Logo を変更

3.13.1 コマンドラインツールで Linux LOGO を変更

Linux システムは起動する時、下記の画像が表示する

対応するするファイルは linux-2.6.32.2/drivers/video/logo/linux_logo_clut224.ppm

画像の logo ファイルを変更する方法はたくさんあり、よく使われるのは netpbm ツールセットである。

゛netpbm゛はコマンドラインのツールで、多数のフォーム画像を変換できる、ここで、png フォームを例として、PNG ファイルを必要の Linux LOGO 画像へ変換する。

仮に、変換されるファイルの名前は linux_logo.png で、まず png 画像は pnm に変換される

pngtopnm linux_logo.png > linux_logo.pnm

png 画像のの色は 224 に制限される

pnmquant 224 linux_logo.pnm > linux_logo_clut224.pnm

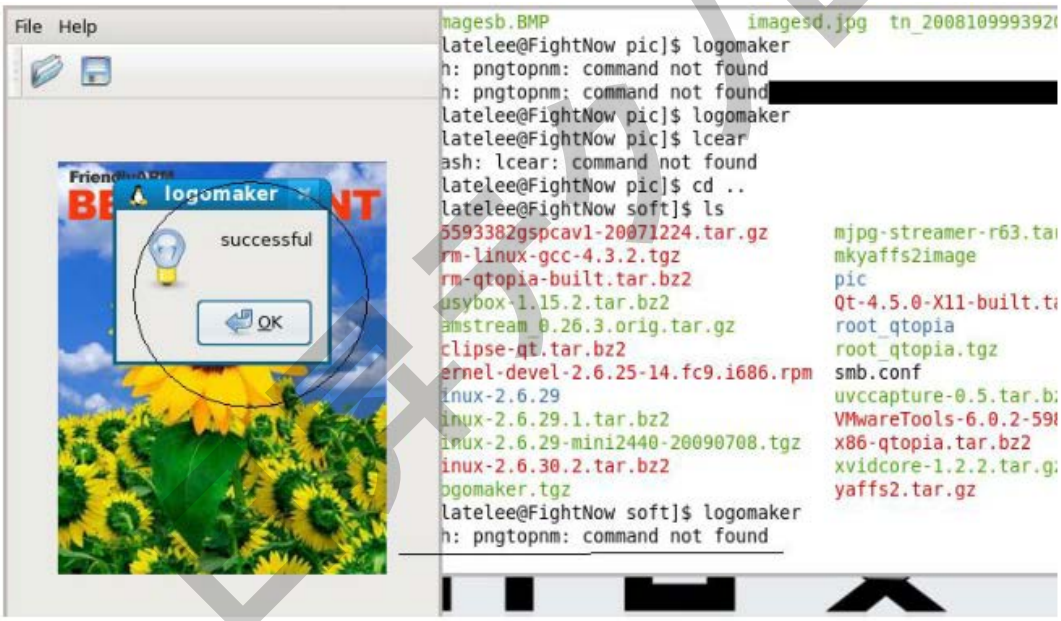
最後に png 画像は必要の Linux LOGO 画像を変換する

pnmtoplainpnm linux_logo_clut224.pnm > linux_logo_clut224.ppm

最後、linux_logo_clut224.ppm は linux-2.6.32.2/drivers/video/logo の対応画像を取り替えるだけ

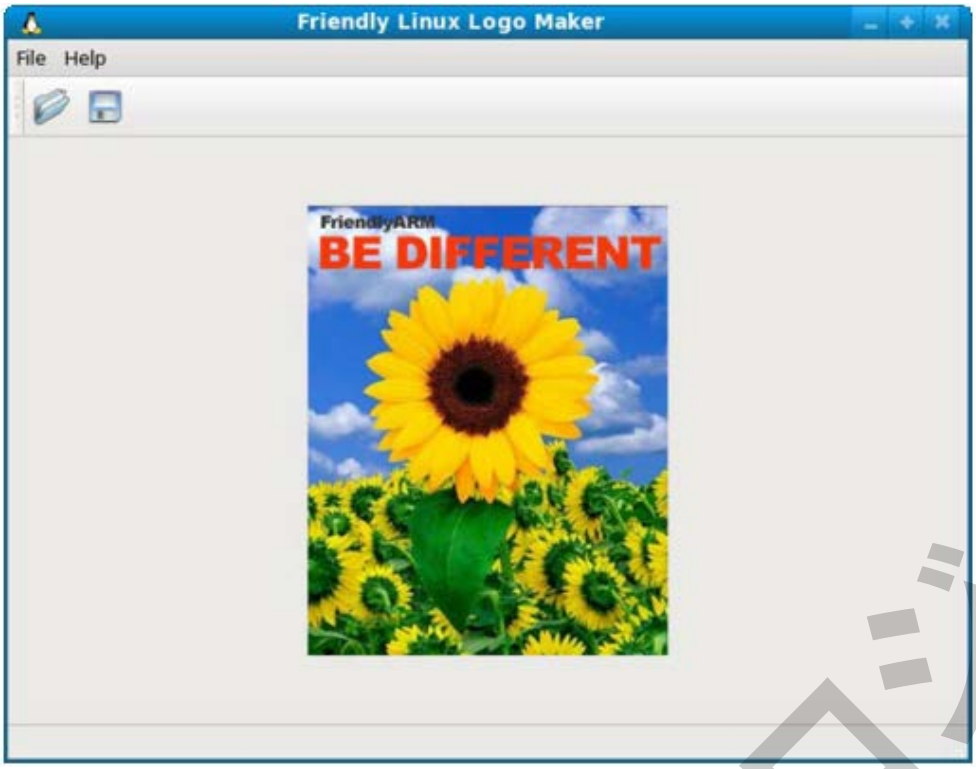
3.13.2 グラフィカル LogoMaker で Linux LOGO を作成

ユーザーに便利に使用させるように LogoMaker を使用し設計し、これは Fedora 9 プラットフォームに基づいて開発し、実はこの最下層が呼び出すのは上述コマンドラインツールである。実行結果は浮動小数点エラーな場合、Fedora9 を使用する時下記図のようなエラーが出る場合、netpbm ツールを正しくインストールされていないと思う（マニュアルの手順に従い Fedora9 プラットフォームをインストール）

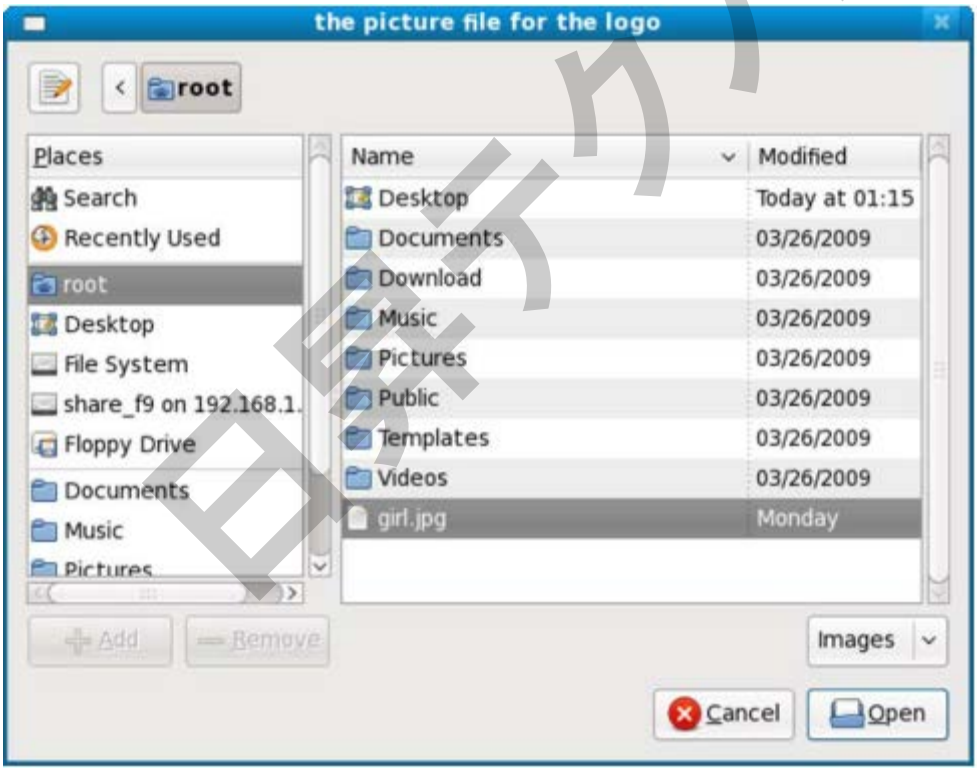


次は Linux LogoMaker の使用手順である（mini2440 ユーザーマニュアルから写し取る）

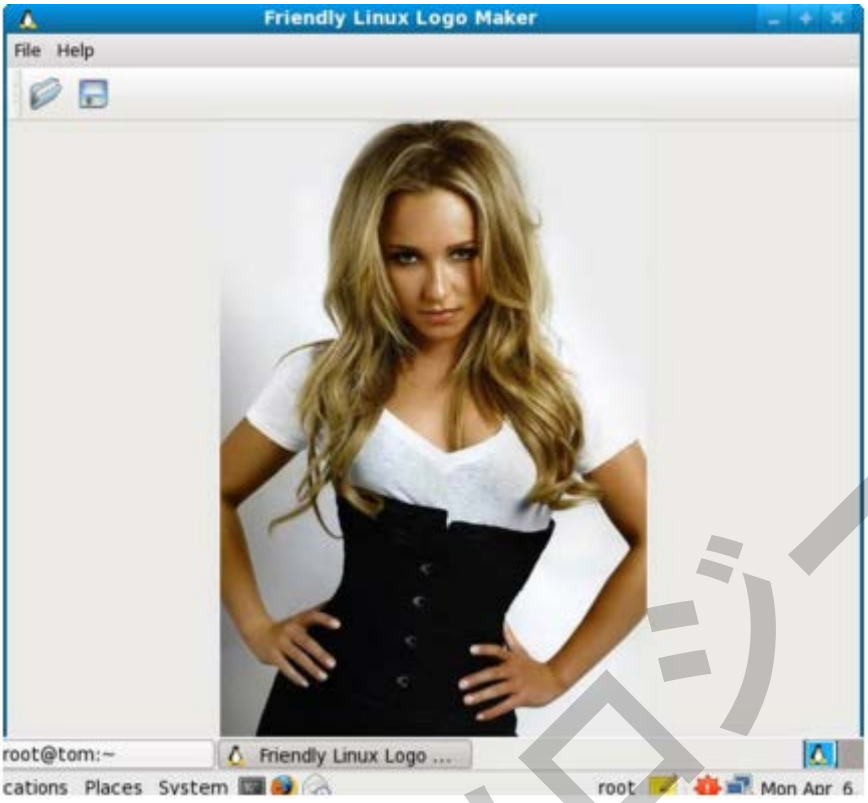
まずユーザーマニュアルに従い logomaker ツールプログラムをインストールし、任意コマンドライン：logomaker を入力し、起動する。花の画像が表示する、下図を参照する



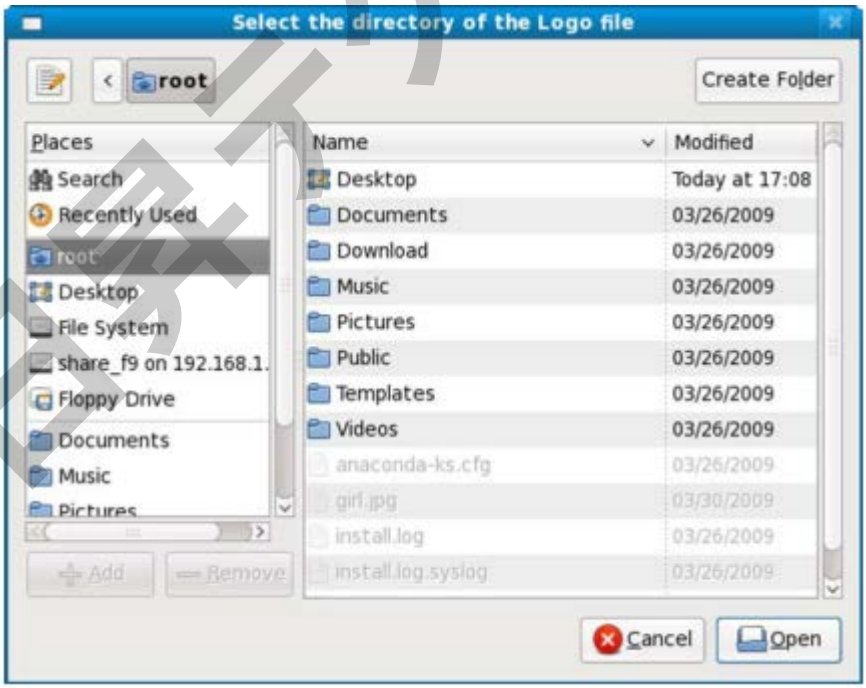
File->Open a picture file…、またはショートカットキーCtrl+O を使用し、画像ファイルをオープン、ファイルウィンドウで画像を選択



画像は logomaker ウィンドウで表示するる、下図を参照する



File->Convert the picture to a Linux Logo File、またはショートカットキーCrtl+C を使用し、ファイルディレクトリ保存、保存ディレクトリを選択し、ファイル名前は linux_logo_clut224.ppm と自動的に保存する。このファイルを linux-2.6.32.2/drivers/video/logo ディレクトリ下の同名ファイルを入れ替える。



LogoMaker の説明は下図を参照する(メニュー゛help->About゛をクリックするだけ、これをオープンする)

3.14　ADC ドライバ追加

3.14.1　S3C2440 の ADC とタッチ･スクリーン･インタフェースについて

Linux-2.6.32.2 カーネルは S3C2440 をサポートする ADC ドライバプログラムを提供しないため、自分で設計必要となり、ドライバはキャラクタデバイスで、drivers/char ディレクトリ下にある、ドライバプログラムのファイルの名前は mini2440_adc.c である

S3C2440 チップには AD 入力とタッチ･スクリーン･インタフェースは共同の A / D コンバータを使用し、2440 データシート第 16 章節を参照する

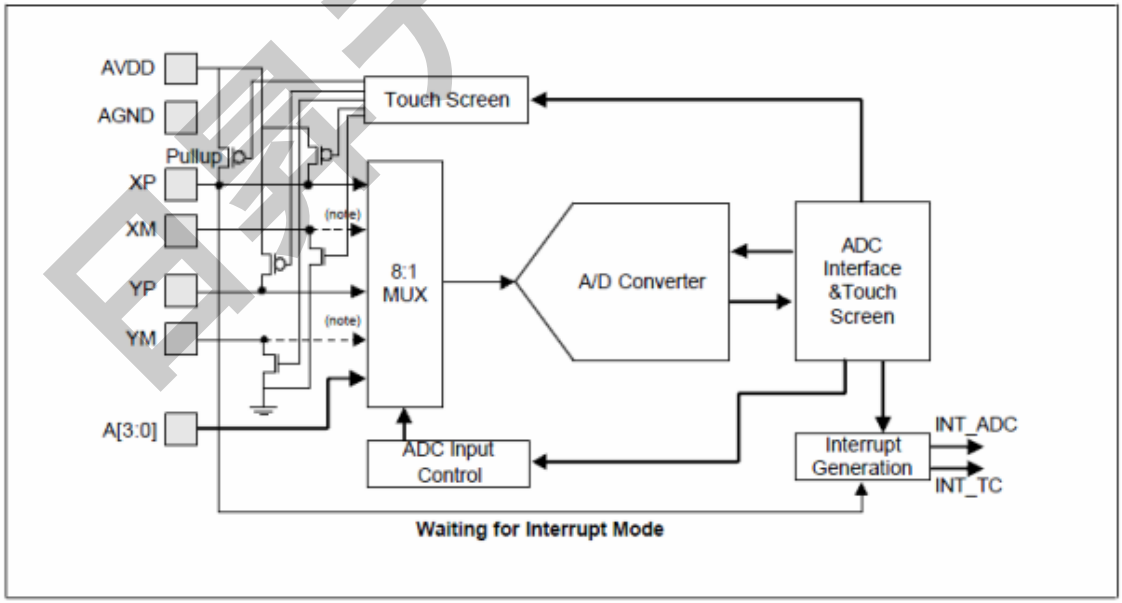


Figure 16-1. ADC and Touch Screen Interface Functional Block Diagram

3.14.2　カーネルに ADC ドライバを追加

ADC ドライバとタッチ･スクリーンドライバは共存させるには、A / D コンバータリソースを共用の問題を

解決する必要がある、ADC ドライバプログラムにグロバル` ADC_LOCK` セマフォ変数を定義し、ADC ドライバプログラムの内容とコメントは下記の通り

```
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/input.h>
#include <linux/init.h>
#include <linux/serio.h>
#include <linux/delay.h>
#include <linux/clk.h>
#include <linux/wait.h>
#include <linux/sched.h>
#include <asm/io.h>
#include <asm/irq.h>
#include <asm/uaccess.h>
#include <mach/regs-clock.h>
#include <plat/regs-timer.h>

#include <plat/regs-adc.h>
#include <mach/regs-gpio.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>

; ネイティブカーネルに含まれていないため、ヘッダファイルは定義します
#include "s3c24xx-adc.h"

#undef DEBUG
//#define DEBUG
#ifdef DEBUG
#define DPRINTK(x...) {printk(__FUNCTION__"(%d): ",__LINE__);printk(##x);}
#else
#define DPRINTK(x...) (void)(0)
#endif

; ADC 変換デバイス名を定義します、/dev/adc に保存します
#define DEVICE_NAME "adc"
static void __iomem *base_addr;
```

```
; ADC デバイス構造を定義します
typedef struct {
    wait_queue_head_t wait;
    int channel;
    int prescale;
}ADC_DEV;

; グローバルセマフォ・ステートメントし、タッチパネルドライバと A / D 変換器を共有します
DECLARE_MUTEX(ADC_LOCK);
; ADC ドライバは A / D 変換器リソースの状態変数の保有状態判断
static int OwnADC = 0;

static ADC_DEV adcdev;
static volatile int ev_adc = 0;
static int adc_data;

static struct clk               *adc_clock;

; ADC レジスタ定義
#define ADCCON              (*(volatile unsigned long *)(base_addr + S3C2410_ADCCON))   //ADC control
#define ADCTSC                      (*(volatile unsigned long *)(base_addr + S3C2410_ADCTSC))    //ADC   touch
screen
control
#define ADCDLY              (*(volatile unsigned long *)(base_addr + S3C2410_ADCDLY))    //ADC start or Interval
Delay
#define ADCDAT0                     (*(volatile unsigned long *)(base_addr + S3C2410_ADCDAT0))   //ADC
conversion
data 0
#define ADCDAT1                     (*(volatile unsigned long *)(base_addr + S3C2410_ADCDAT1))   //ADC
conversion
data 1
#define ADCUPDN                      (*(volatile unsigned long *)(base_addr + 0x14)) //Stylus Up/Down interrupt
status

#define PRESCALE_DIS    (0 << 14)
#define PRESCALE_EN     (1 << 14)
#define PRSCVL(x)                   ((x) << 6)
#define ADC_INPUT(x)    ((x) << 3)
```

```
;“AD 入力オン”マクロを定義します
#define START_ADC_AIN(ch, prescale) ¥
    do{ ¥
        ADCCON = PRESCALE_EN | PRSCVL(prescale) | ADC_INPUT((ch)) ; ¥
        ADCCON |= ADC_START; ¥
    }while(0)

;ADC 割り込みハンドラ関数
static irqreturn_t adcdone_int_handler(int irq, void *dev_id)
{
    ; ADC ドライバが“A/D 変換器”ソースがある場合、 ADC レジスタから変換結果を読み取ります
    if (OwnADC) {
        adc_data = ADCDAT0 & 0x3ff;
        ev_adc = 1;

        wake_up_interruptible(&adcdev.wait);
    }

    return IRQ_HANDLED;
}

;ADC   read 関数、ユーザーレイヤ/アブリレイヤデバイスの（read）関数
static ssize_t s3c2410_adc_read(struct file *filp, char *buffer, size_t count, loff_t *ppos)
{
    char str[20];
    int value;
    size_t len;

    ;“A/D 変換器”  ソース有効性判断
    if (down_trylock(&ADC_LOCK) == 0) {
        OwnADC = 1; //“A/D 変換器”  ソースステータスを有効に標識
        START_ADC_AIN(adcdev.channel, adcdev.prescale); //変換開始
        wait_event_interruptible(adcdev.wait, ev_adc); //端末を介して変換の結果を待ちます

        ev_adc = 0;

        DPRINTK("AIN[%d] = 0x%04x, %d¥n", adcdev.channel, adc_data, ADCCON & 0x80 ? 1:0);
```

```
        ;変換結果を value に与えます。ユーザーレイヤ/アブリレイヤに伝送準備します
        value = adc_data;

        ; “A/D 変換器” ソースリリース
        OwnADC = 0;
        up(&ADC_LOCK);
    } else {
        ;“A/D 変換器” ソースなし、値は゛-1゛となり
        value = -1;
    }

    len = sprintf(str, "%d¥n", value);
    if (count >= len) {
        ; 変換結果をユーザーレイヤ/アブリレイヤに伝送します
        int r = copy_to_user(buffer, str, len);
        return r ? r : len;
    } else {
        return -EINVAL;
    }
}

;ADC デバイス関数をオープン、ユーザー・モード・アプリケーションの open と対応します
static int s3c2410_adc_open(struct inode *inode, struct file *filp)
{
    ; 割り込みキュー初期化
    init_waitqueue_head(&(adcdev.wait));

    ; デフォルト・チャネルは"0"
    adcdev.channel=0;
    adcdev.prescale=0xff;

    DPRINTK( "adc opened¥n");
    return 0;
}

static int s3c2410_adc_release(struct inode *inode, struct file *filp)
{
    DPRINTK( "adc closed¥n");
```

```
    return 0;
}

static int s3c2410_adc_release(struct inode *inode, struct file *filp)
{
    DPRINTK( "adc closed¥n");
    return 0;
}

static struct file_operations dev_fops = {
    owner:          THIS_MODULE,
    open:           s3c2410_adc_open,
    read:s3c2410_adc_read,
    release: s3c2410_adc_release,
};

static struct miscdevice misc = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DEVICE_NAME,
    .fops = &dev_fops,
};

static int __init dev_init(void)
{
    int ret;

    base_addr=ioremap(S3C2410_PA_ADC,0x20);
    if (base_addr == NULL) {
        printk(KERN_ERR "Failed to remap register block¥n");
        return -ENOMEM;
    }

    adc_clock = clk_get(NULL, "adc");
    if (!adc_clock) {
        printk(KERN_ERR "failed to get adc clock source¥n");
        return -ENOENT;
    }
    clk_enable(adc_clock);
```

```
    /* normal ADC */
    ADCTSC = 0;

    ;割り込み登録
    ret = request_irq(IRQ_ADC, adcdone_int_handler, IRQF_SHARED, DEVICE_NAME, &adcdev);
    if (ret) {
            iounmap(base_addr);
            return ret;
    }

    ;デバイス登録
    ret = misc_register(&misc);

    printk (DEVICE_NAME"¥tinitialized¥n");
    return ret;
}

static void __exit dev_exit(void)
{
    ;割り込みリリース
    free_irq(IRQ_ADC, &adcdev);
    iounmap(base_addr);

    if (adc_clock) {
            clk_disable(adc_clock);
            clk_put(adc_clock);
            adc_clock = NULL;
    }

    misc_deregister(&misc);
}
; タッチスクリーンドライバに使用できるように、セマフォ"ADC_LOCK"をエクスポートします
EXPORT_SYMBOL(ADC_LOCK);
module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
```

上記のドライバプログラムには、簡単なヘッダファイル头文件ﾞs3c24xx-adc.hﾞを含め、これは

drivers/char ディレクトリ下にある、内容は下記の通り

```
#ifndef _S3C2410_ADC_H_
#define _S3C2410_ADC_H_

#define ADC_WRITE(ch, prescale) ((ch)<<16|(prescale))

#define ADC_WRITE_GETCH(data) (((data)>>16)&0x7)
#define ADC_WRITE_GETPRE(data) ((data)&0xff)

#endif /* _S3C2410_ADC_H_ */
```

その後、drivers/char/Makefile ファイルをオープンし、大体 114 行に ADC ドライバプログラム目標モジュールを追加する

```
obj-$(CONFIG_JS_RTC)                    += js-rtc.o
js-rtc-y = rtc.o

obj-$(CONFIG_MINI2440_ADC) += mini2440_adc.o

# Files generated that shall be removed upon make clean
clean-files := consolemap_deftbl.c defkeymap.c
```

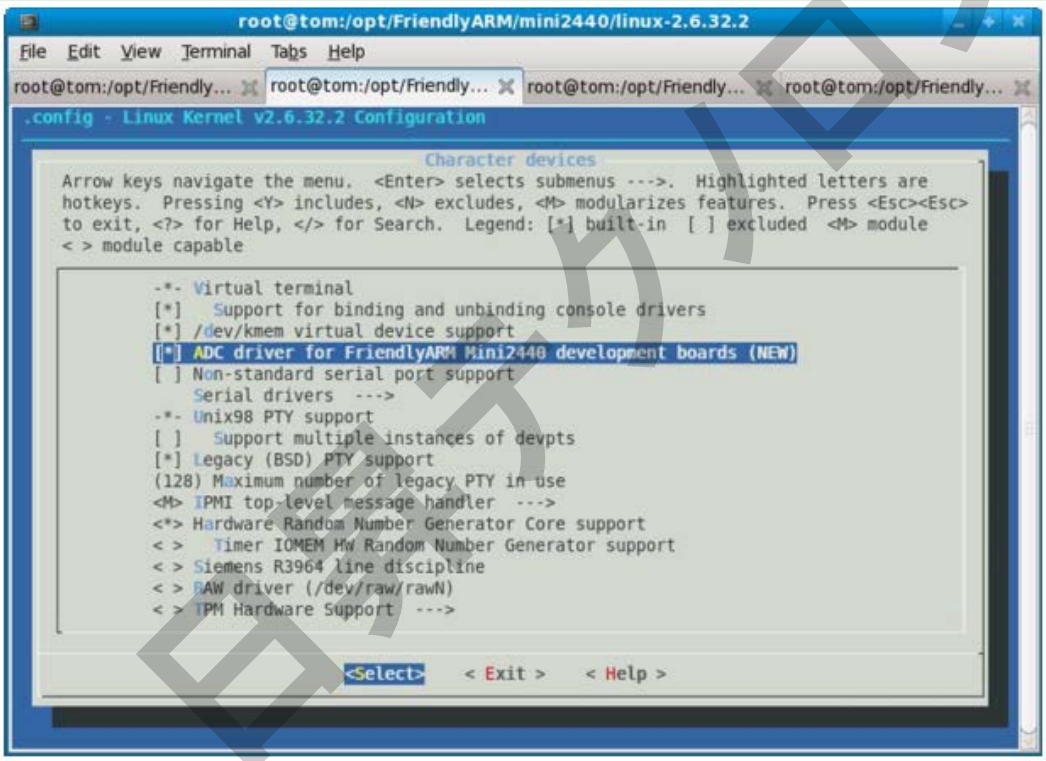
次に drivers/char/Kconfig をオープンし、ADC ドライバ設定オプションを追加する

```
config DEVKMEM
        bool "/dev/kmem virtual device support"
        default y
        help
          Say Y here if you want to support the /dev/kmem device. The
          /dev/kmem device is rarely used, but can be used for certain
          kind of kernel debugging operations.
          When in doubt, say "N".

config MINI2440_ADC
        bool "ADC driver for FriendlyARM Mini2440 development boards"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
        help
          this is ADC driver for FriendlyARM Mini2440 development boards
          Notes: the touch-screen-driver required this option
```

```
config BFIN_JTAG_COMM
        tristate "Blackfin JTAG Communication"
        depends on BLACKFIN
        help
```

カーネルに ADC ドライバを追加し、カーネルソースコードディレクトリのコマンドライン：make menuconfig を実行し、次のサブメニューを順に選択し、先に追加した ADC ドライバ設定オプションを探すスペースバーを押して、ADC 設定オプションを選択する

```
Device Drivers --->
 Character devices --->
```



その後、終了し、設定保存、コマンドライン：make zImage を実行し、arch/arm/boot/zImage を生成し、supervivi の゛k゛コマンドを使用し、開発ボードにプログラミング

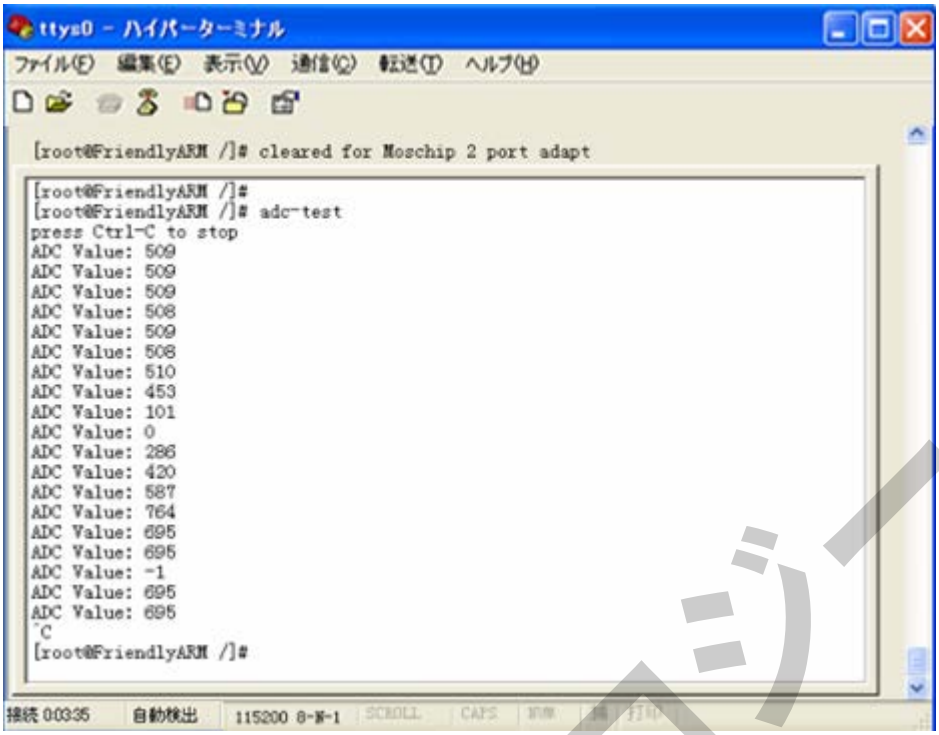
3.14.3 ADC テストプログラム

ここで既存ファイルシステムを使用する。中に adc-test コマンドがあり、ソースコード：

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
```

```
#include <fcntl.h>
#include <linux/fs.h>
#include <errno.h>
#include <string.h>

int main(void)
{
    fprintf(stderr, "press Ctrl-C to stop¥n");
    int fd = open("/dev/adc", 0);
    if (fd < 0) {
        perror("open ADC device:");
        return 1;
    }
    for(;;) {
        char buffer[30];
        int len = read(fd, buffer, sizeof buffer -1);
        if (len > 0) {
            buffer[len] = '¥0';
            int value = -1;
            sscanf(buffer, "%d", &value);
            printf("ADC Value: %d¥n", value);
        } else {
            perror("read ADC device:");
            return 1;
        }
        usleep(500* 1000);
    }

    close(fd);
}
```

〝adc-test〟テストプログラムは既にファイルシステムに統合されるため、開発ボードのコマンドラインターミナルに adc-test を入力し、開発ボードの W1 可変抵抗を回転すると、ADC 変換の結果も変化すると見つけて、タッチスクリーンを押すと、〝-1〟を出力し、ドライバプログラムの設定と同じ、下図を参照する



3.15　タッチスクリーンドライバを追加（詳しい原理分析を付き）

3.15.1　カーネルにタッチスクリーンドライバプログラムを追加

Linux-2.6.32.2 カーネルは S3C2440 をサポートするタッチスクリーンドライバを含まないため、自分で s3c2410_ts.c を設計し、linux-src/drivers/input/touchscreen ディレクトリ下にあり、s3c2410_ts.c ファイルを追加、次の内容をコピーする

```
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/input.h>
#include <linux/init.h>
#include <linux/serio.h>
#include <linux/delay.h>
#include <linux/platform_device.h>
#include <linux/clk.h>
#include <linux/gpio.h>
#include <asm/io.h>
#include <asm/irq.h>

#include <plat/regs-adc.h>
#include <mach/regs-gpio.h>
```

```
/* For ts.dev.id.version */
#define S3C2410TSVERSION	0x0101

#define WAIT4INT(x)	(((x)<<8) | ¥
		S3C2410_ADCTSC_YM_SEN|S3C2410_ADCTSC_YP_SEN|S3C2410_ADCTSC_XP_SEN|¥
		S3C2410_ADCTSC_XY_PST(3))

#define AUTOPST	(S3C2410_ADCTSC_YM_SEN | S3C2410_ADCTSC_YP_SEN
|S3C2410_ADCTSC_XP_SEN | ¥
		S3C2410_ADCTSC_AUTO_PST | S3C2410_ADCTSC_XY_PST(0))

static char *s3c2410ts_name = "s3c2410 TouchScreen";

static  struct input_dev *dev;
static  long xp;
static  long yp;
static  int count;

extern struct semaphore ADC_LOCK;
static int OwnADC = 0;

static void __iomem *base_addr;

static inline void s3c2410_ts_connect(void)
{
    s3c2410_gpio_cfgpin(S3C2410_GPG(12), S3C2410_GPG12_XMON);
    s3c2410_gpio_cfgpin(S3C2410_GPG(13), S3C2410_GPG13_nXPON);
    s3c2410_gpio_cfgpin(S3C2410_GPG(14), S3C2410_GPG14_YMON);
    s3c2410_gpio_cfgpin(S3C2410_GPG(15), S3C2410_GPG15_nYPON);
}

static void touch_timer_fire(unsigned long data)
{
    unsigned long data0;
    unsigned long data1;
    int updown;
```

```
    data0 = ioread32(base_addr+S3C2410_ADCDAT0);
    data1 = ioread32(base_addr+S3C2410_ADCDAT1);

    updown=(!(data0&S3C2410_ADCDAT0_UPDOWN))&&(!(data1 & S3C2410_ADCDAT0_UPDOWN));

    if (updown) {
        if (count != 0) {
            long tmp;

            tmp = xp;
            xp = yp;
            yp = tmp;

                        xp >>= 2;
                        yp >>= 2;

            input_report_abs(dev, ABS_X, xp);
            input_report_abs(dev, ABS_Y, yp);

            input_report_key(dev, BTN_TOUCH, 1);
            input_report_abs(dev, ABS_PRESSURE, 1);
            input_sync(dev);
        }
        xp = 0;
        yp = 0;
        count = 0;

        iowrite32(S3C2410_ADCTSC_PULL_UP_DISABLE|AUTOPST, ase_addr+S3C2410_ADCTSC);
        iowrite32(ioread32(base_addr+S3C2410_ADCCON)|S3C2410_ADCCON_ENABLE_START,
base_addr+S3C2410_ADCCON);
    } else {
        count = 0;

        input_report_key(dev, BTN_TOUCH, 0);
        input_report_abs(dev, ABS_PRESSURE, 0);
        input_sync(dev);
```

```
            iowrite32(WAIT4INT(0), base_addr+S3C2410_ADCTSC);
            if (OwnADC) {
                OwnADC = 0;
                up(&ADC_LOCK);
            }
        }
    }

    static struct timer_list touch_timer =
            TIMER_INITIALIZER(touch_timer_fire, 0, 0);

    static irqreturn_t stylus_updown(int irq, void *dev_id)
    {
        unsigned long data0;
        unsigned long data1;
        int updown;

        if (down_trylock(&ADC_LOCK) == 0) {
            OwnADC = 1;
            data0 = ioread32(base_addr+S3C2410_ADCDAT0);
            data1 = ioread32(base_addr+S3C2410_ADCDAT1);

    updown = (!(data0 & S3C2410_ADCDAT0_UPDOWN)) && (!(data1 &
S3C2410_ADCDAT0_UPDOWN));

            if (updown) {
                touch_timer_fire(0);
            } else {
                OwnADC = 0;
                up(&ADC_LOCK);
            }
        }

        return IRQ_HANDLED;
    }

    static irqreturn_t stylus_action(int irq, void *dev_id)
```

```
{
    unsigned long data0;
    unsigned long data1;

    if (OwnADC) {
        data0 = ioread32(base_addr+S3C2410_ADCDAT0);
        data1 = ioread32(base_addr+S3C2410_ADCDAT1);

        xp += data0 & S3C2410_ADCDAT0_XPDATA_MASK;
        yp += data1 & S3C2410_ADCDAT1_YPDATA_MASK;
        count++;

        if (count < (1<<2)) {
            iowrite32(S3C2410_ADCTSC_PULL_UP_DISABLE | AUTOPST,
base_addr+S3C2410_ADCTSC);
            iowrite32(ioread32(base_addr+S3C2410_ADCCON)|S3C2410_ADCCON_ENABLE_START,
base_addr+S3C2410_ADCCON);
        } else {
            mod_timer(&touch_timer, jiffies+1);
            iowrite32(WAIT4INT(1), base_addr+S3C2410_ADCTSC);
        }
    }

    return IRQ_HANDLED;
}

static struct clk *adc_clock;

static int __init s3c2410ts_init(void)
{
    struct input_dev *input_dev;

    adc_clock = clk_get(NULL, "adc");
    if (!adc_clock) {
        printk(KERN_ERR "failed to get adc clock source¥n");
        return -ENOENT;
    }
    clk_enable(adc_clock);
```

```
base_addr=ioremap(S3C2410_PA_ADC,0x20);
if (base_addr == NULL) {
	printk(KERN_ERR "Failed to remap register block¥n");
	return -ENOMEM;
}

/* Configure GPIOs */
s3c2410_ts_connect();

iowrite32(S3C2410_ADCCON_PRSCEN | S3C2410_ADCCON_PRSCVL(0xFF),¥
		base_addr+S3C2410_ADCCON);
iowrite32(0xffff, base_addr+S3C2410_ADCDLY);
iowrite32(WAIT4INT(0), base_addr+S3C2410_ADCTSC);

/* Initialise input stuff */
input_dev = input_allocate_device();

if (!input_dev) {
	printk(KERN_ERR "Unable to allocate the input device !!¥n");
	return -ENOMEM;
}

dev = input_dev;
dev->evbit[0] = BIT(EV_SYN) | BIT(EV_KEY) | BIT(EV_ABS);
dev->keybit[BITS_TO_LONGS(BTN_TOUCH)] = BIT(BTN_TOUCH);
input_set_abs_params(dev, ABS_X, 0, 0x3FF, 0, 0);
input_set_abs_params(dev, ABS_Y, 0, 0x3FF, 0, 0);
input_set_abs_params(dev, ABS_PRESSURE, 0, 1, 0, 0);

dev->name = s3c2410ts_name;
dev->id.bustype = BUS_RS232;
dev->id.vendor = 0xDEAD;
dev->id.product = 0xBEEF;
dev->id.version = S3C2410TSVERSION;

/* Get irqs */
if (request_irq(IRQ_ADC, stylus_action, IRQF_SHARED|IRQF_SAMPLE_RANDOM,
```

```
            "s3c2410_action", dev)) {
            printk(KERN_ERR "s3c2410_ts.c: Could not allocate ts IRQ_ADC !¥n");
            iounmap(base_addr);
            return -EIO;
    }
    if (request_irq(IRQ_TC, stylus_updown, IRQF_SAMPLE_RANDOM,
                    "s3c2410_action", dev)) {
            printk(KERN_ERR "s3c2410_ts.c: Could not allocate ts IRQ_TC !¥n");
            iounmap(base_addr);
            return -EIO;
    }

    printk(KERN_INFO "%s successfully loaded¥n", s3c2410ts_name);

    /* All went ok, so register to the input system */
    input_register_device(dev);

    return 0;
}

static void __exit s3c2410ts_exit(void)
{
    disable_irq(IRQ_ADC);
    disable_irq(IRQ_TC);
    free_irq(IRQ_TC,dev);
    free_irq(IRQ_ADC,dev);

    if (adc_clock) {
            clk_disable(adc_clock);
            clk_put(adc_clock);
            adc_clock = NULL;
    }

    input_unregister_device(dev);
    iounmap(base_addr);
}

module_init(s3c2410ts_init);
module_exit(s3c2410ts_exit);
```

その後 linux-2.6.32.2/drivers/input/touchscreen/Makefile に該当ソースコードのターゲットモジュールを追加し、図の赤い部分のように示す

```
obj-$(CONFIG_TOUCHSCREEN_WM97XX_ZYLONITE)       += zylonite-wm97xx.o
obj-$(CONFIG_TOUCHSCREEN_W90X900)               += w90p910_ts.o
obj-$(CONFIG_TOUCHSCREEN_PCAP)     += pcap_ts.o
obj-$(CONFIG_TOUCHSCREEN_S3C2410) += s3c2410_ts.o
```

次に linux-2.6.32.2/drivers/input/touchscreen/Kconfig ファイルをオープンし、次の赤い部分を追加

```
menuconfig INPUT_TOUCHSCREEN
        bool "Touchscreens"
        help
          Say Y here, and a list of supported touchscreens will be displayed.
          This option doesn't affect the kernel.

          If unsure, say Y.

if INPUT_TOUCHSCREEN

config TOUCHSCREEN_S3C2410
        tristate "Samsung S3C2410 touchscreen input driver"
        depends on MACH_MINI2440 && INPUT && INPUT_TOUCHSCREEN && MINI2440_ADC
        help
          Say Y here if you have the s3c2410 touchscreen.

          If unsure, say N.

          To compile this driver as a module, choose M here: the
          module will be called s3c2410_ts.

config TOUCHSCREEN_ADS7846
        tristate "ADS7846/TSC2046 and ADS7843 based touchscreens"
        depends on SPI_MASTER
        depends on HWMON = n || HWMON
        help
```

し、カーネル設定に mini2440 のタッチスクリーンドライバオプションを追加する

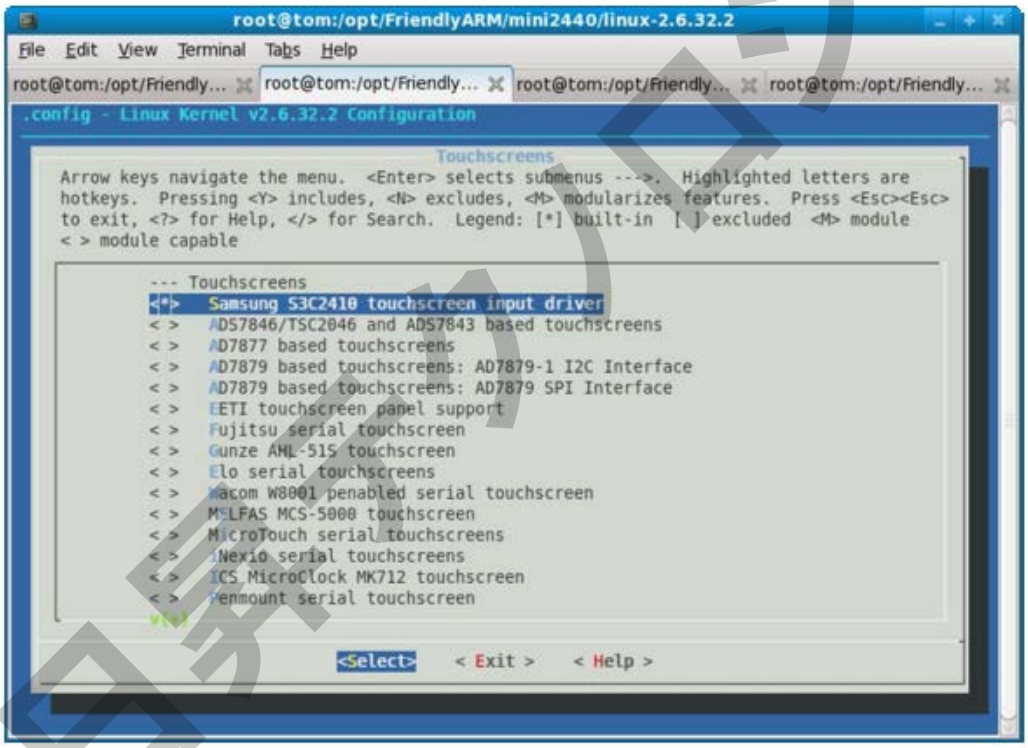
ここまで、カーネルにタッチスクリーンドライバの追加完了

3.15.2 カーネルを設定し、コンパイルし、タッチスクリーンドライバをテスト

コマンドライン：make menuconfig を実行し、次のサブメニューを順に選択し、先追加したタッチスクリーンドライバを探す

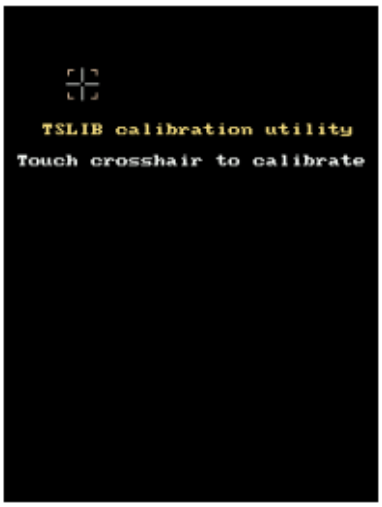
```
Device Drivers --->
    Input device support --->
        [*] Touchscreens --->
```

スペースバーを押して、タッチスクリーンドライバ設定オプションを選択する



完了後、上記のカーネル設定を保存し、コマンドライン：make zImage を入力し、arch/arm/boot/zImage ファイルを生成する。supervivi の゛k゛コマンドを使用し、開発ボードにプログラミングする。

デフォルトのファイルシステム root_qtopia を使用し、スクリーンにキャリブレーション･インタフェースを確認できる。



スクリーンに従いタッチペンで、十字型のクロスポイントをクリックする、qtopia システムに入る

3.15.3　タッチスクリーンドライバ原理の詳しい説明

下記のネット文章はタッチスクリーンドライバ原理について詳しく分析する。

mini2440 タッチスクリーンドライバプログラム分析

By JeefJiang July、8th、2009

これは mini2440 ドライバ分析シリーズの第三弾の文章で、本文は三つ部分に分けられ、第一部分はハードウェア知識を説明し、タッチスクリーンの原理と SCC2440 SOC のタッチスクリーンはどのように操作するかと含める。第二部分は入力デバイスのサブシステムのフレームワークを分析し、そして該当のコード分析を行う。第三部分は上述の原理を使用し、mini2440 タッチスクリーンドライバを分析する。第四部分はテストとチェックを紹介する

1、必要のハードウェアの準備知識

1.1、抵抗式タッチスクリーン操作原理

タッチスクリーンはディスプレイの表面に付着し、ディスプレイと合わせて使用する、タッチポイントはスクリーンにある座標位置を計測できれば、ディスプレイに対応する座標点の表示内容またはアイコンより、タッチ者の意図を分かれる。タッチスクリーンは、その技術原理に基づいて 5 つタイプに分けられる：ベクトル·圧力感知型、抵抗型、キャパシタンス型、赤外線型、弾性表面波型。その中で、組み込みシステムで抵抗型タッチスクリーンはより多く使われる、抵抗型タッチスクリーンは 4 層の透明複合フィルムスクリーンで、一番下はガラスやプレキシガラスで構成するベース層で、一番上は外表面が硬化処理を経て、滑らかで、スクラッチのことを防ぐプラスチック層で、真ん中には 2 つの金属導電層であり、それぞれベース層の上とプラスチック層の裏にあり、二つの導電層の間にこれらを分ける多く小さな透明隔離点がある。

指がスクリーンに触れる時、二つの導電層は、タッチ·ポイントに接触している

タッチスクリーンの 2 つの金属導電層はタッチスクリーンの二つ作業面であり、作業面毎の両端にエポキシ銀ペーストが塗りつけられ、該当動作面の一対電極と呼ばれ、一つ作業面の電極に上へ電圧を印加すると、該当作業面には均一な連続パラレル電圧分布を形成する。X 方向の電極に上へある確定電圧を印加し、Y 方向の電極に上へある確定電圧を印加しない場合、X パラレル電圧フィールドに、接触点の電圧値は Y+(または Y-)電極に反映され、Y +電極の接地電圧の大きさの計測を通じて、接触点の X 座標値を取得できる。

同様に Y 電極に上へある電圧を印加し、X 電極に上へある電圧を印加しない場合、X+電極の電圧の計測を通じて接触点の XY 座標値を分かる。抵抗式タッチスクリーンは四線式と五線式があり、四線式タッチスクリーンの X 作業面と Y 作業面は二つ導電層にそれぞれかけられ、四つリードはタッチスクリーンの X 電極対と Y 電極対にそれぞれ接続し、五線式タッチスクリーンの X 作業面と Y 作業面はガラス下部の導電層にかけられるが、動作の時、まだタイムシェアリングに電圧を印かし、即ち二つ方向の電圧フィールドに同一作業面にタイムシェアリングに動作させる、外側導電層がただ導体と電圧測定用電極として使われる。従って、五線式タッチスクリーンは五つリードが必要である

1.2　S3C2440 の中のタッチ･スクリーン･インタフェース

SOC S3C2440 のタッチ･スクリーン･インタフェースは ADC インタフェースと結び付ける、詳しくは次の通り

コンバージョン率：PCLK=50MHz の時、デバイダは 49 に設定されると、10 ビットのコンバージョン計算は下記の通り：

When the GCLK frequency is 50MHz and the prescaler value is 49、

A/D converter freq. = 50MHz/(49+1) = 1MHz

Conversion time = 1/(1MHz / 5cycles) = 1/200KHz = 5 us

This A/D converter was designed to operate at maximum 2.5MHz clock、 so the conversion rate can go up to 500 KSPS.

タッチスクリーン･インタフェースのモードは下記のタイプがあり

普通 ADC コンバージョンモード

独立 X/Y 位置コンバージョンモード

自動 X/Y 位置コンバージョンモード

割り込み待ちモード

我々は主にタッチ･スクリーン･インタフェースの割り込み待ちモードと自動 X/Y 位置コンバージョンモードを受信する（ドライバプログラムに使われる）

自動コンバージョンモードの操作プロセスは下記の通りであり：タッチ･スクリーンは X、Y のタッチ位置の自動コンバージョンを制御し、コンバージョンが完了してから、データはそれぞれレジスタ ADCDAT0 と ADCDAT1 に保存され、また、INT_ADC 割り込みが起き、コンバージョン完了を知らせる

割り込み待ちモード

Touch Screen Controller generates interrupt (INT_TC) signal when the Stylus is down. Waiting for Interrupt Modesetting value is rADCTSC=0xd3; // XP_PU、 XP_Dis、 XM_Dis、 YP_Dis、 YM_En.

タッチした後、タッチスクリーンコントローラは INT_TC 割り込みを生成し、四つピンの設定は下記の通りであり

ピン	XP	XM	YP	YM
状態	PULL UP/XP Disable	Disable（初期値はそうである）	Disable	Enable
設定	1	0	1	1

割り込みが起きてから X/Y の位置データは独立 X/Y 位置コンバージョンモードと自動 X/Y 位置コンバージョンモードを選択し、読み取る。自動 X/Y 位置コンバージョンモードを使用し、読み取る時、既に設定されたレジスタを変更する必要があり、既存の基礎に 3C2410_ADCTSC_PULL_UP_DISABLE |

S3C2410_ADCTSC_AUTO_PST |S3C2410_ADCTSC_XY_PST(0)

データがコンバージョンしてから、割り込みも起きる

2、入力サブシステムのモデル分析

2.1 全体的なフレームワーク

入力サブシステムは三つ部分を含まれる：デバイスドライバ、入力コア、イベントハンドラ

第一部分は各バスに接続された入力デバイスドライバで、SOC にこのバスは仮想バス platformbus を使える、これらの役割は最下層のハードウェア入力が統一イベント･タイプに変更されるのことで、入力コア（Input core）に報告する

第二部分：入力コアの役割は下記の通りであり

（1）input_register_device() used to を呼び出し、デバイスを追加し、input_unregister_device()を呼び出し、デバイスを取り外す（次はタッチスクリーンドライブと結び付け、説明する）

（2）/PROC のもとに該当のデバイス情報を生成される、次の例はこのようにである：

/proc/bus/input/devices showing a USB mouse:

I: Bus=0003 Vendor=046d Product=c002 Version=0120

N: Name="Logitech USB-PS/2 Mouse M-BA47"

P: Phys=usb-00:01.2-2.2/input0

H: Handlers=mouse0 event2

B: EV=7

B: KEY=f0000 0 0 0 0 0 0 0 0

B: REL=103

（3）イベント･ハンドラーはイベントを処理すると知らせ

第三部分はイベント･ハンドラーである

入力サブシステムは必要なプロセッサを含まれ、例えばマウス、キーボード、joystick、タッチスクリーン、event handler と呼ばれる汎用のプロセッサ（カーネルファイル evdev.C に対する）注意を頂きたい点は、カーネルバージョンの発展に従い、event handler はより多い異なるハードウェアの入力イベントを処理する時に使われる。Linux2.6.29 バージョンに残る特定デバイスイベント処理はただマウスと joystick があり、これはますます多い入力デバイスが event handler を通じてユーザースペースとやり取りを行われ、イベント処理層の主な役割はユーザースペースとやり取りを行うことである。Linux はユーザースペースにすべてのデバイスがファイルとして処理でき、普通のドライバプログラムに fops インタフェースが提供され、/dev のもとに該当のデバイスファイル nod を生成し、入力サブシステムのドライバのなか、これらの動作は全部イベントハンドラ層に完成される。evdev.C の関連コードを見る

```
static int __init evdev_init(void)
{
return input_register_handler(&evdev_handler);
}
```

上記のはモジュールの登録プロセスである、システム初期化の時呼び出される。

初期化プロセスは簡単、evdev_handler に保存される：

```
static struct input_handler evdev_handler = {
.event        = evdev_event、
```

```
.connect    = evdev_connect、
.disconnect    = evdev_disconnect、
.fops        = &evdev_fops、
.minor        = EVDEV_MINOR_BASE、
.name        = "evdev"、
.id_table    = evdev_ids、
};
```

まず connect 関数の下記のコードを見て：

```
snprintf(evdev->name、 sizeof(evdev->name)、 "event%d"、 minor);
evdev = kzalloc(sizeof(struct evdev)、 GFP_KERNEL);
evdev->handle.dev = input_get_device(dev);
evdev->handle.name = evdev->name;
dev_set_name(&evdev->dev、 evdev->name);
evdev->dev.devt = MKDEV(INPUT_MAJOR、 EVDEV_MINOR_BASE + minor);
evdev->dev.class = &input_class;    evdev
->dev.parent = &dev->dev;
evdev->dev.release = evdev_free;
device_initialize(&evdev->dev);
error = device_add(&evdev->dev);
```

注：上記の部分は/sys/device/viture/input/input0/event0 ディレクトリに生成し、event の中に dev の属性ファイルがあり、デバイスファイルのデバイス番号を保存し、このように udev は該当属性ファイルを読み取り、デバイス番号を取得できる、これによって/dev ディレクトリにデバイスノード/dev/event0 を作成する。次は evdev_fops メンバーを見る

```
static const struct file_operations evdev_fops = {
.owner        = THIS_MODULE、
.read        = evdev_read、
.write        = evdev_write、
.poll        = evdev_poll、
.open        = evdev_open、
.release    = evdev_release、    .unlocked_ioctl    = evdev_ioctl、
#ifdef CONFIG_COMPAT
.compat_ioctl    = evdev_ioctl_compat、
#endif
.fasync        = evdev_fasync、    .flush        = evdev_flush
}
```

これはデバイスがユーザースペース向けのインタフェースで、デバイスへのアクセスになり、その中に evdev_ioctl は多いコマンドを提供し、関連のコマンドは《Using the Input Subsystem、 Part II》を参照する

3　mini2440 のタッチスクリーンドライバ

3.1 初期化

```
static int __init s3c2410ts_init(void)
{
struct input_dev *input_dev;

adc_clock = clk_get(NULL、 "adc");
if (!adc_clock) {
printk(KERN_ERR "failed to get adc clock source¥n");
return -ENOENT;
}
clk_enable(adc_clock);
```

//クロックを取得し、APB BUS の周辺デバイスをマウントし、クロック制御が必要で、ADC と類似のデバイスである。

```
base_addr=ioremap(S3C2410_PA_ADC、0x20);
```

I/O メモリは直接にアクセスできず、マッピングされる必要があり、I/O に仮想アドレスを割り当てし、これらの仮想アドレスは__iomem で説明し、直接にアクセスできない、専門の関数を使用する必要があり、例えば iowrite32

```
if (base_addr == NULL) {
printk(KERN_ERR "Failed to remap register block¥n");
return -ENOMEM;
}

/* Configure GPIOs */
s3c2410_ts_connect();
iowrite32(S3C2410_ADCCON_PRSCEN | S3C2410_ADCCON_PRSCVL(0xFF)、¥
base_addr+S3C2410_ADCCON);// プリ分割・オン/分割係数・設定する
iowrite32(0xffff、 base_addr+S3C2410_ADCDLY);//ADC 遅延時間を設定し、割り込み待ちモードのもとに、INT_TC を起こる間隔時間を表示する
iowrite32(WAIT4INT(0)、 base_addr+S3C2410_ADCTSC);
```

割り込み待ちモードに従い TSC を設定する

次は入力デバイスの登録

```
/* Initialise input stuff */
input_dev = input_allocate_device();
```

//allocate memory for new input device、入力デバイスにメモリを割り当てる際に使用され、また、入力デバイスの初期化設定をする

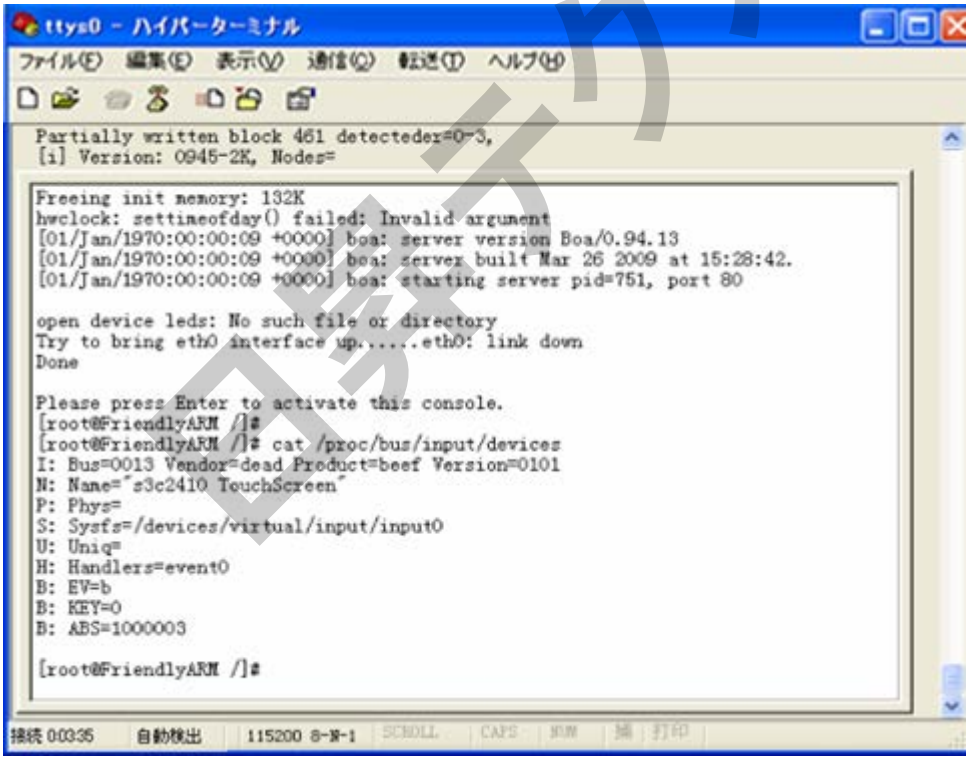
```
if (!input_dev) {
printk(KERN_ERR "Unable to allocate the input device !!¥n");
return -ENOMEM;
}
```

```
dev = input_dev;
dev->evbit[0] = BIT(EV_SYN) | BIT(EV_KEY) | BIT(EV_ABS);
//イベントタイプを設定する
dev->keybit[BITS_TO_LONGS(BTN_TOUCH)] = BIT(BTN_TOUCH);
input_set_abs_params(dev、 ABS_X、 0、 0x3FF、 0、 0);
input_set_abs_params(dev、 ABS_Y、 0、 0x3FF、 0、 0);
input_set_abs_params(dev、 ABS_PRESSURE、 0、 1、 0、 0);
```

上記の四つプログラム分はイベントタイプを設定する code で、まず、イベントタイプを説明し、よく使われるイベントタイプ EV_KEY、EV_MOSSE、 EV_ABS(タッチスクリーンのような絶対座標イベントを受信するに使用される)、タイプ毎のイベントは異なるタイプのコード code がある、

例えば ABS_X、ABS_Y、これらのコードは該当の value がある

```
dev->name = s3c2410ts_name;
dev->id.bustype = BUS_RS232;
dev->id.vendor = 0xDEAD;
dev->id.product = 0xBEEF;
dev->id.version = S3C2410TSVERSION;
//上記は入力デバイスの名前と id であり、これらの情報は入力デバイスの ID 情報であり、
cat /proc/bus/input/devices、
```




```
/* Get irqs */
if (request_irq(IRQ_ADC、 stylus_action、 IRQF_SAMPLE_RANDOM、
"s3c2410_action"、 dev)) {
printk(KERN_ERR "s3c2410_ts.c: Could not allocate ts IRQ_ADC !¥n");
```

```
iounmap(base_addr);
return -EIO;
}
if (request_irq(IRQ_TC、 stylus_updown、 IRQF_SAMPLE_RANDOM、
"s3c2410_action"、 dev)) {
printk(KERN_ERR "s3c2410_ts.c: Could not allocate ts IRQ_TC !¥n");
iounmap(base_addr);
return -EIO;
}

printk(KERN_INFO "%s successfully loaded¥n"、 s3c2410ts_name);

/* All went ok、 so register to the input system */
input_register_device(dev);
//デバイスの基本情報と基本機能は設定完了し、直接登録できる
    return 0;
}
```

割り込み処理

stylus_action と stylus_updown 二つ割り込みハンドラ関数、ペンでタッチするする時、stylus_updown に入る、

```
static irqreturn_t stylus_updown(int irq、 void *dev_id)
{
unsigned long data0;
unsigned long data1;
int updown;
```

//注：タッチスクリーンドライバモジュールに、ADC_LOCK がある。そしてその機能は常に一つドライバプログラムのみが ADC の割り込みラインを使用と保証する。mini2440adc モジュールも ADC を使用するため、ロックがある限り、起動 ADC に入れる。注：LDD3 で信号はスリープの原因で ISR に適合しないが、down_trylock は例外で、スリープに入れない。

```
if (down_trylock(&ADC_LOCK) == 0) {
OwnADC = 1;
data0 = ioread32(base_addr+S3C2410_ADCDAT0);
data1 = ioread32(base_addr+S3C2410_ADCDAT1);

updown = (!(data0 & S3C2410_ADCDAT0_UPDOWN)) && (!(data1 &
S3C2410_ADCDAT0_UPDOWN));

if (updown) {//means down
touch_timer_fire(0);//タイマー関数、普通関数として呼び出し、ADC を起動する
```

```
} else {
OwnADC = 0;
up(&ADC_LOCK);//
}
}

return IRQ_HANDLED;
}
static void touch_timer_fire(unsigned long data)
{
unsigned long data0;
unsigned long data1;
int updown;
data0 = ioread32(base_addr+S3C2410_ADCDAT0);
data1 = ioread32(base_addr+S3C2410_ADCDAT1);
updown = (!(data0 & S3C2410_ADCDAT0_UPDOWN)) && (!(data1 &
S3C2410_ADCDAT0_UPDOWN));

if (updown) {//means down
//四回コンバージョンしてからイベント報告を行う
if (count != 0) {
long tmp;

tmp = xp;
xp = yp;
yp = tmp;
//使用するクリーンは 240*320 であり、プロトのスクリーンの X、Y 軸を変換する。
xp >>= 2;
yp >>= 2;
input_report_abs(dev、 ABS_X、 xp);
input_report_abs(dev、 ABS_Y、 yp);
//デバイス X、Y 値
input_report_key(dev、 BTN_TOUCH、 1);
input_report_abs(dev、 ABS_PRESSURE、 1);
input_sync(dev);
//タッチスクリーンイベントの情報、次の報告を間隔に使用される
}
xp = 0;
yp = 0;
```

```
count = 0;

iowrite32(S3C2410_ADCTSC_PULL_UP_DISABLE | AUTOPST、
base_addr+S3C2410_ADCTSC);
iowrite32(ioread32(base_addr+S3C2410_ADCCON) |
S3C2410_ADCCON_ENABLE_START、 base_addr+S3C2410_ADCCON);
//ADC を起動できず、或いは ACD は四回にコンバージョンした後 ADC を起動する

}       else {
//Up 状態な場合、報告を提出し、スクリーンをタッチ待ちモードに入る
count = 0;

input_report_key(dev、 BTN_TOUCH、 0);
input_report_abs(dev、 ABS_PRESSURE、 0);
input_sync(dev);

iowrite32(WAIT4INT(0)、 base_addr+S3C2410_ADCTSC);
if (OwnADC) {
OwnADC = 0;
up(&ADC_LOCK);
}
}
}
static irqreturn_t stylus_action(int irq、 void *dev_id)
{
unsigned long data0;
unsigned long data1;

if (OwnADC) {
data0 = ioread32(base_addr+S3C2410_ADCDAT0);
data1 = ioread32(base_addr+S3C2410_ADCDAT1);

xp += data0 & S3C2410_ADCDAT0_XPDATA_MASK;
yp += data1 & S3C2410_ADCDAT1_YPDATA_MASK;
count++;
//データを読み取る
if (count < (1<<2)) {//四回より小さくなる場合、再度にコンバージョンを起動する
iowrite32(S3C2410_ADCTSC_PULL_UP_DISABLE | AUTOPST、
base_addr+S3C2410_ADCTSC);
```

```
iowrite32(ioread32(base_addr+S3C2410_ADCCON) |
S3C2410_ADCCON_ENABLE_START、 base_addr+S3C2410_ADCCON);
} else {//四回を超える場合、1ms を待ってからデータの報告を行う

mod_timer(&touch_timer、 jiffies+1);
iowrite32(WAIT4INT(1)、 base_addr+S3C2410_ADCTSC);

}
}

return IRQ_HANDLED;
}
```

全体からのコンバージョンのプロセスを説明する：

（1）タッチスクリーンは、タッチを感じた場合、updown ISR に入る、ADC_LOCK を取得する場合、touch_timer_fire を呼び出し、ADC を起動する

（2）ADC コンバージョン、四回より小さくなる場合、コンバージョンし続き、四回が完了してから、SysTick のタイマーを起動し、ADC を停止し、即ちこの期間内に、ADC は停止の状態である

（3）このようにスクリーンのジッターを防げる。

（4）一つ SysTick 周期が届き、タッチスクリーンはまだタッチ状態な場合、コンバージョンデータを報告し、また ADC を再度起動し、（2）を繰り返す

（5）タッチペンをリリースされた場合、リリースイベントを報告し、また、タッチスクリーンは割り込み待ちモードに戻ります。

4 テストとキャリブレーション

アプリケーションの書きについて《Using the Input Subsystem、 Part II》を参照する 、入力デバイスの API を説明し、タッチスクリーンをキャリブレーションする時、タッチスクリーンの座標に LCD の座標と対応させ、この対応はマッピングする必要があり、このマッピングプロセスはキャリブレーションであり、我々は、線形アルゴリズムのマッピング方法を提供する。

3.16 USB 外部デバイス設定

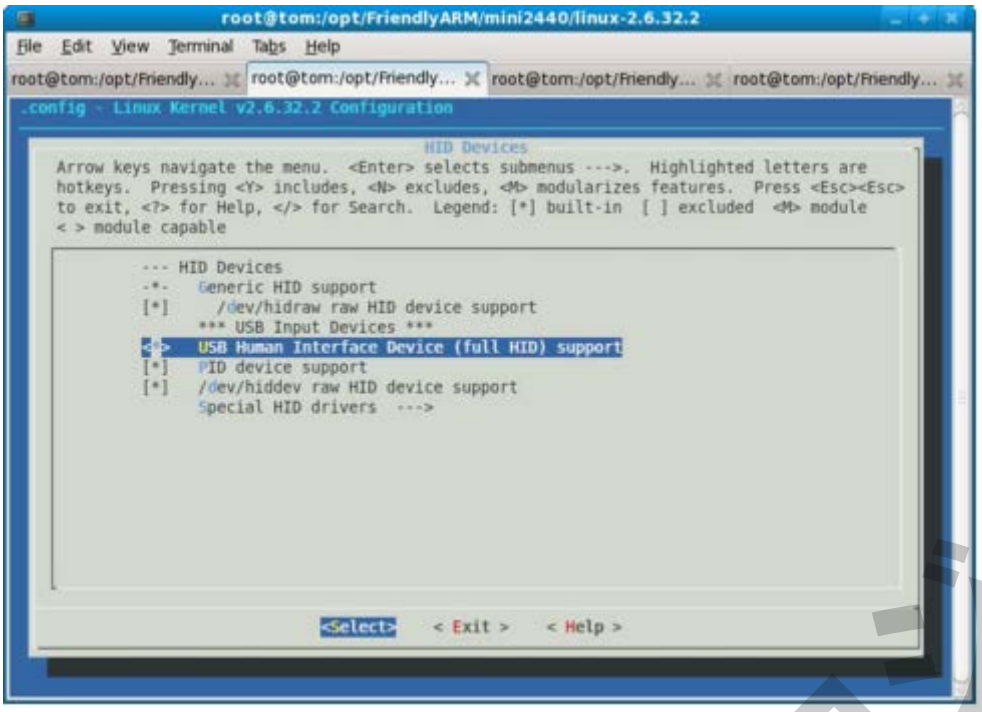
Linux-2.6.32.2 カーネルは USB 外部デバイスに対するサポートは豊かで、S3C2440 の USB Host ドライバサポートを含めるため、カーネルを設定するだけ、次は各種 USB 外部デバイスの詳しい設定手順である

3.16.1 USB キーボード、スキャナ、マウスの設定とテスト

カーネルソースコードディレクトリのターミナルに make menuconfig を入力し、次のサブメニューオプションを順に選択する

```
Device Drivers  --->
  [*] HID Devices  --->
```

カーネル設定メニューが出る

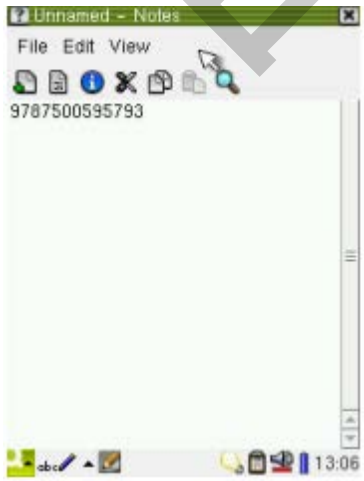


スペースキーを押して、`USB Human Interface Device (full HID) support`を選択し、このように、USB ボタンボードとマウスオプションを設定する

注：ここの設定オプションが対応するカーネルソースコードディレクトリは：linux-2.6.32.2/drivers/hid/usbhid であり、その中の USB キーボードとバーコードスキャナの原理は同じ

3.16.2 USB キーボード、スキャナ、マウスをテスト

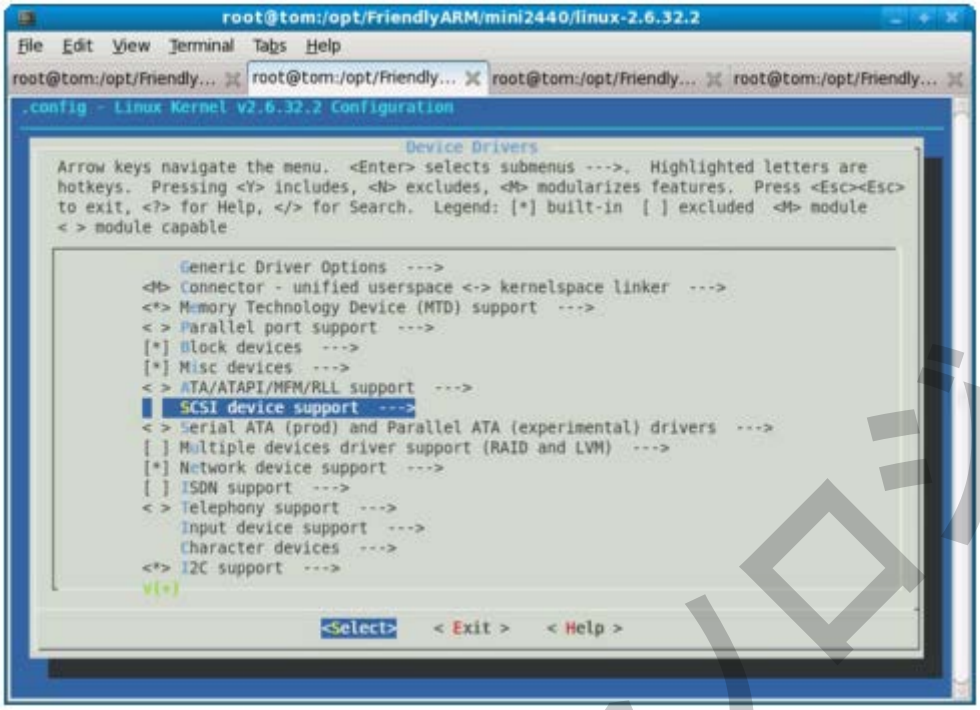
カーネルソースコードルートディレクトリに移動し、make zImage を実行し、生成した新カーネルを開発ボードにプログラミングし、ファイルシステム root_qtopia を使用し、テストする。USB キーボード、マウスとタッチスクリーンを同時にサポートする、また、ホットスワップをサポートし、使用しやすい。前の手順にタッチスクリーンのクリックを通じて既に qtopia グラフィックスシステムに入ったため、ここで直接 USB HUB を探し、USB マウスとキーボード、USB バーコードスキャナなどに接続できる。マウスを使用してアプリケーションを見つける、例えば qtopia の"付箋"機能、起動するとキーボードで、各英語文字を入力でき、USB バーコードスキャナで、バーコードを直接にスキャンし、入力できる。下図を参照する



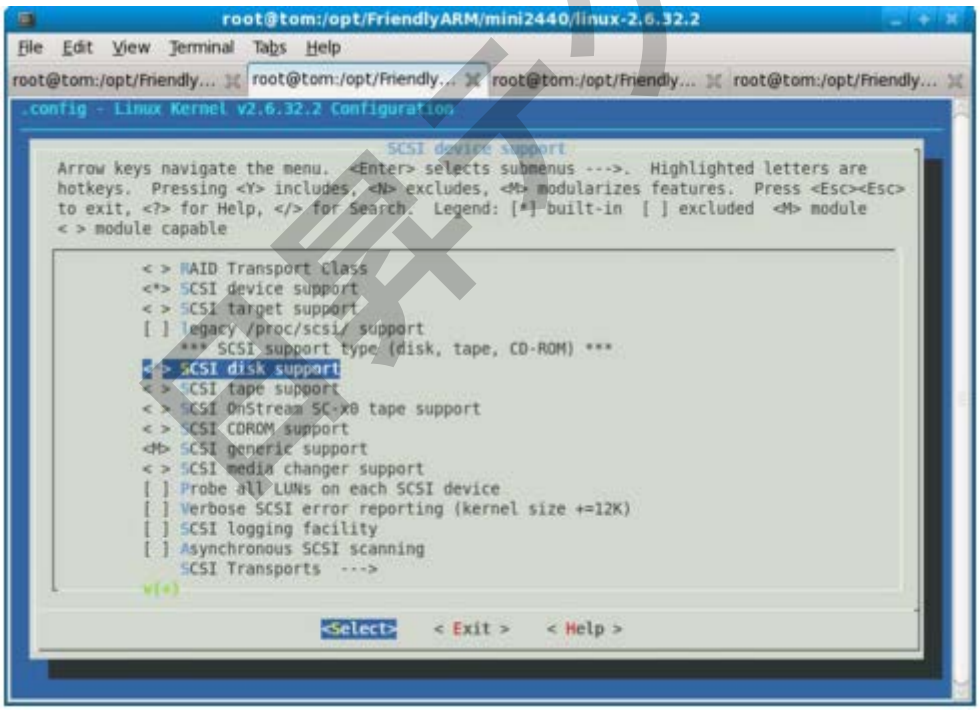
3.16.3 USB メモリードライバを設定

USB メモリードライバは SCSI コマンドを使用するため、まず SCSI サポートを追加する

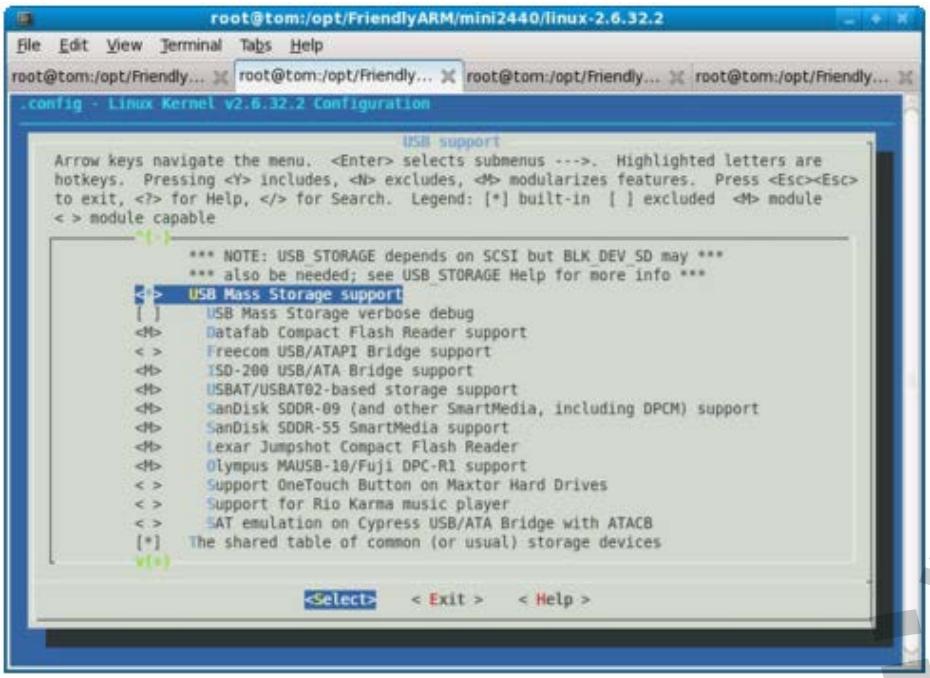
Device Drivers メニューに SCSI device support を選択し、Enter で入る



次スペースキーを押し、オプションを選択する



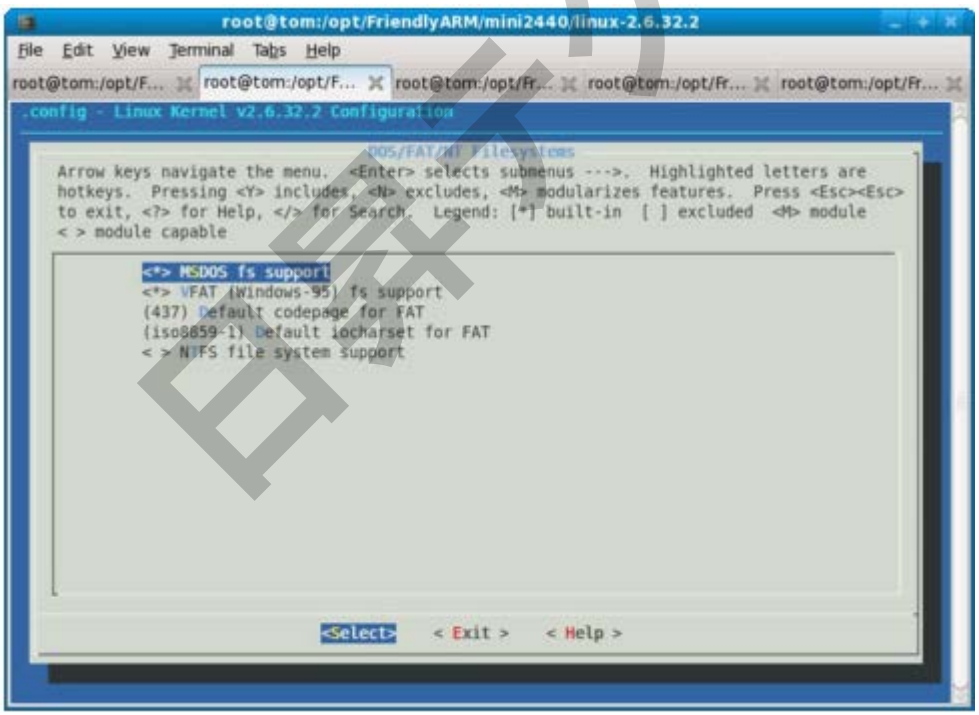
Device Drivers メニューに戻り、USB support を選択し、Enter ボタンを押して USB support メニューに入る、`〈*〉USB Mass Storage support`を探して、選択する、下図を参照する



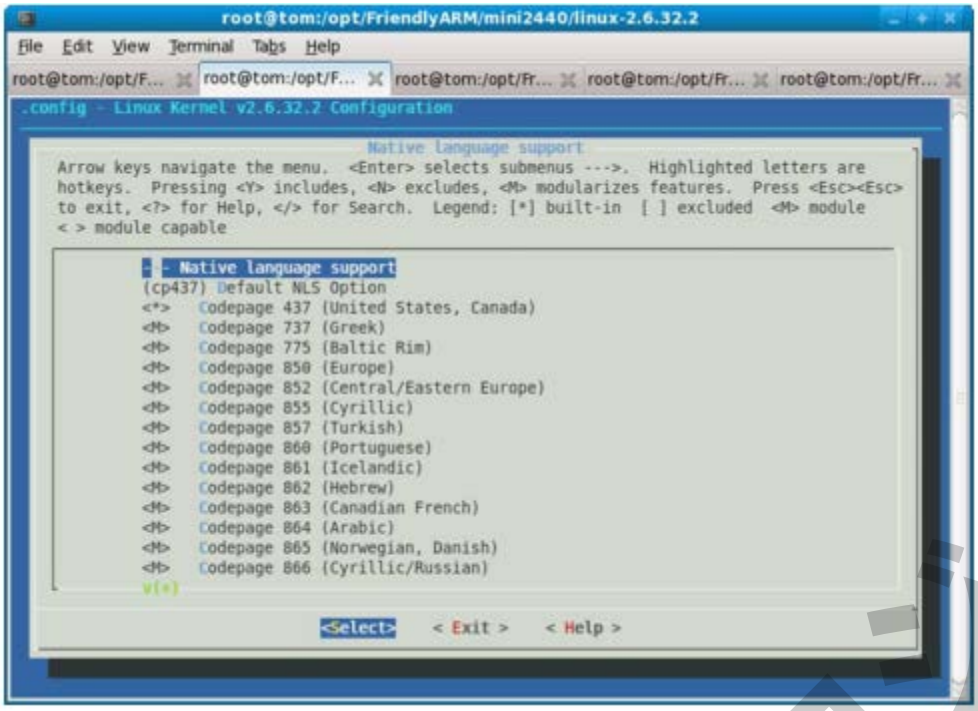
汎用 USB メモリードライバなどのモバイルメモリは多数 FAT/FAT32 フォーマットを使用し、だから FAT32 ファイルシステムのサポートを追加する必要があり、カーネル設定メインメニューのもとに次のメニューオプションを順に選択

```
File systems    --->
  DOS/FAT/NT Filesystems    --->
```

FAT32 ファイルシステム設定サブメニューに入り、図にように選択する



〝File systems〞メニューで〝-*- Native language support ---〉〞を選択し、日本語、中国語と英語のエンコーディングをサポートできる。下図を参照する



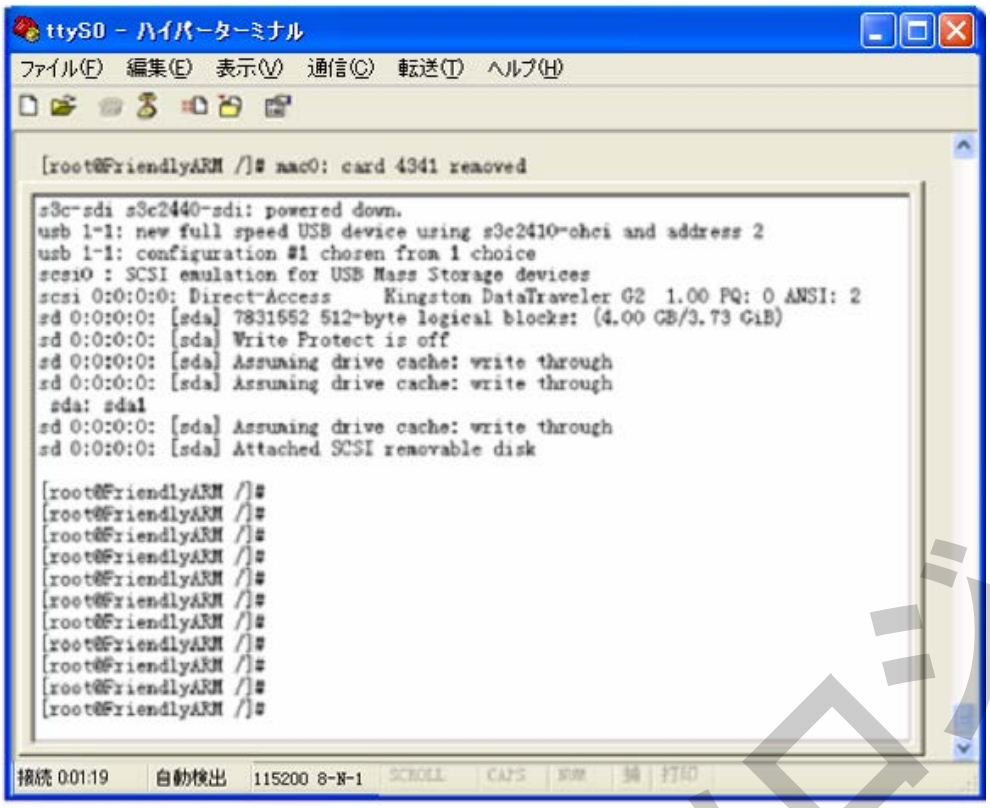
次のコーディングを選択する

<*>	Codepage 437 (United States, Canada)
<*>	NLS ISO 8859-1 (Latin 1; Western European Languages)
<*>	NLS UTF-8

終了、設定保存。

3.16.4 USB メモリードライバをテスト

上述の手順に続き、カーネルソースコードルートディレクトリから make zImage を実行し、生成した新カーネルは開発ボードにプログラミングし、USB メモリードライバを先ず差し込み、システムが起動してからコマンドラインのコンソールに入り、この時の USB メモリードライバに次の情報が出る。



この時、USB メモリードライバは既に開発ボードの/udisk ディレクトリに自動的にマウントされ、同時に Qtopia システムにタスクバーにアイコンを確認できる、下図を参照する

USB メモリードライバにあるすべてファイルは`**ドキュメント**`グループに表示されるが、ディレクトリ名前は表示しない、ファイルは多すぎると、リストはかなり数がある。

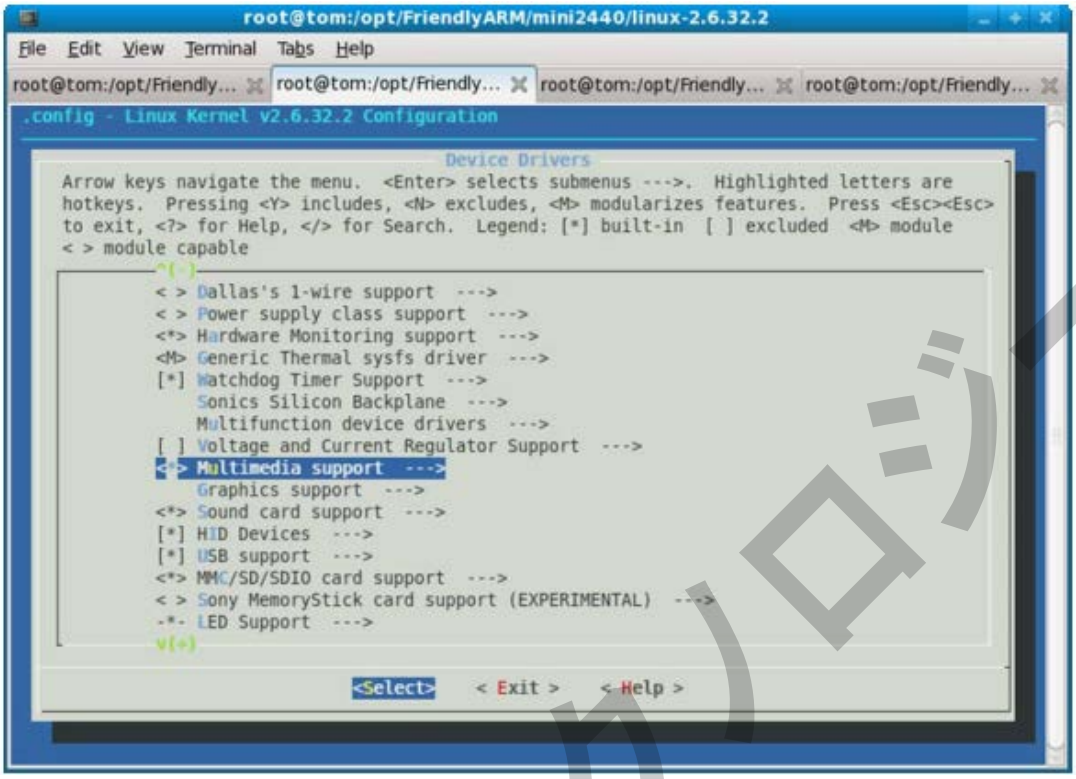
説明：Qtopia は Qtopia 2.2.0 プラグインを通じて USB メモリードライバを自動マウントできる、MMC/SD カードまたは USB メモリードライバの第一のパーティションとして識別する。フォーマットは VFAT/FAT32/FAT16 などで、USB メモリーや SD カードを識別出来る場合、VFAT/FAT32/FAT16 のフォーマットを先ず確認する必要がある。

3.16.5 USB カメラの設定とテスト

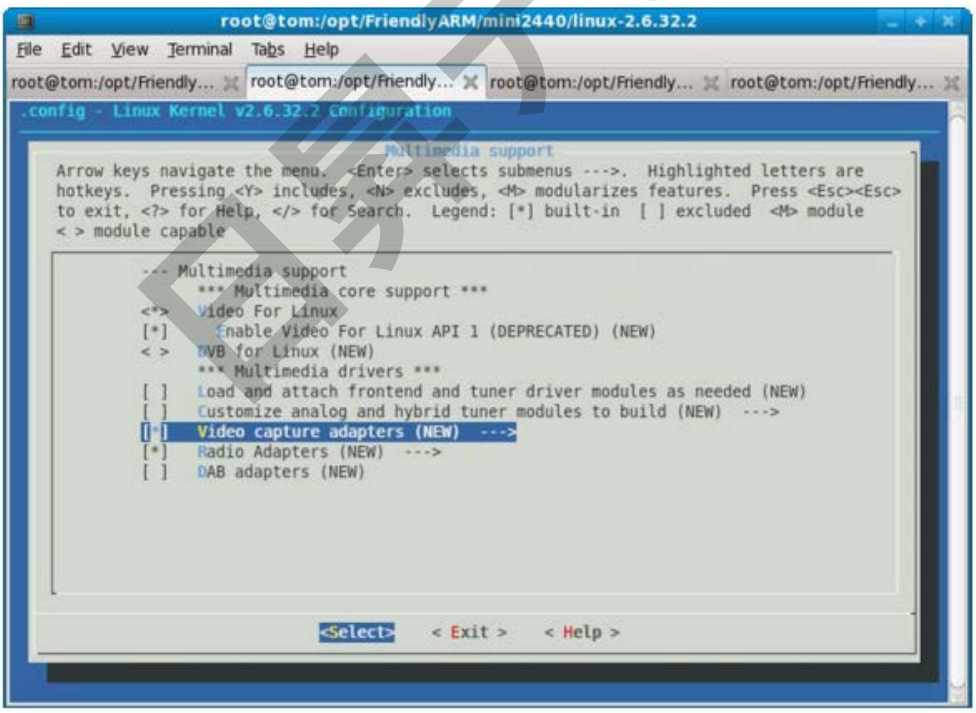
Linux カーネルバージョンをよく更新される理由はドライバサポートを追加するのが主な一つ原因である。

ディレクトリ Linux-2.6.32.2 は既にほとんど USB カメラドライバをサポートするが、メーカーの USB カメラドライバが上層に提供するインターフェースは異なり、カーネルがカメラをサポートしても、普通の USB カメラアプリケーションは認識できない。次はカーネルに USB カメラを設定する手順である

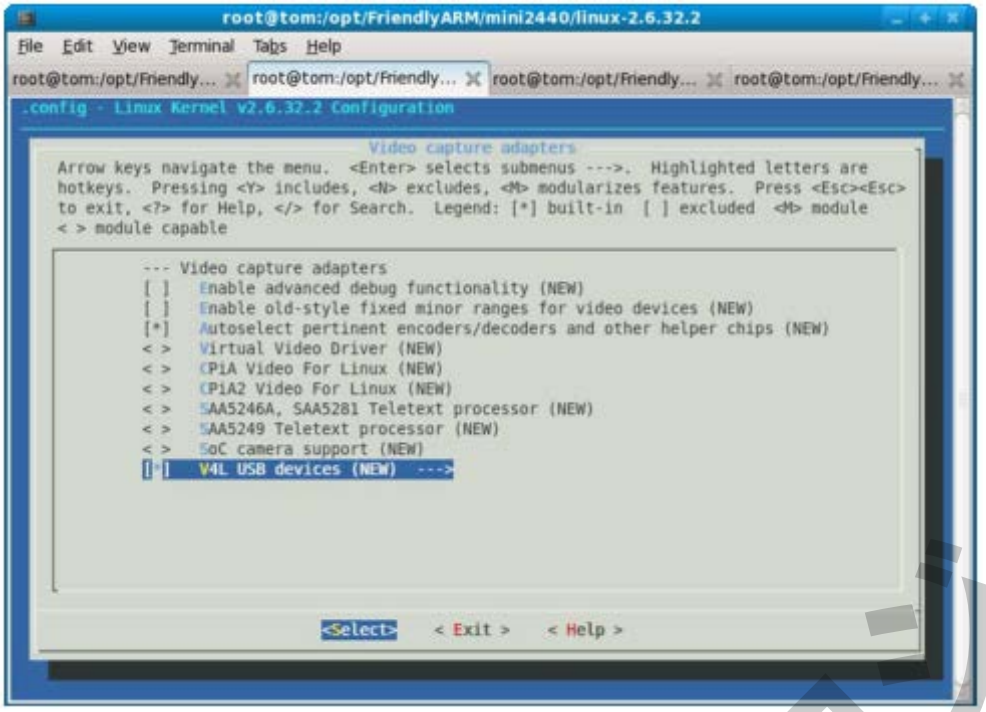
Device Drivers メニューに Multimedia devices を選択し、Enter で入る



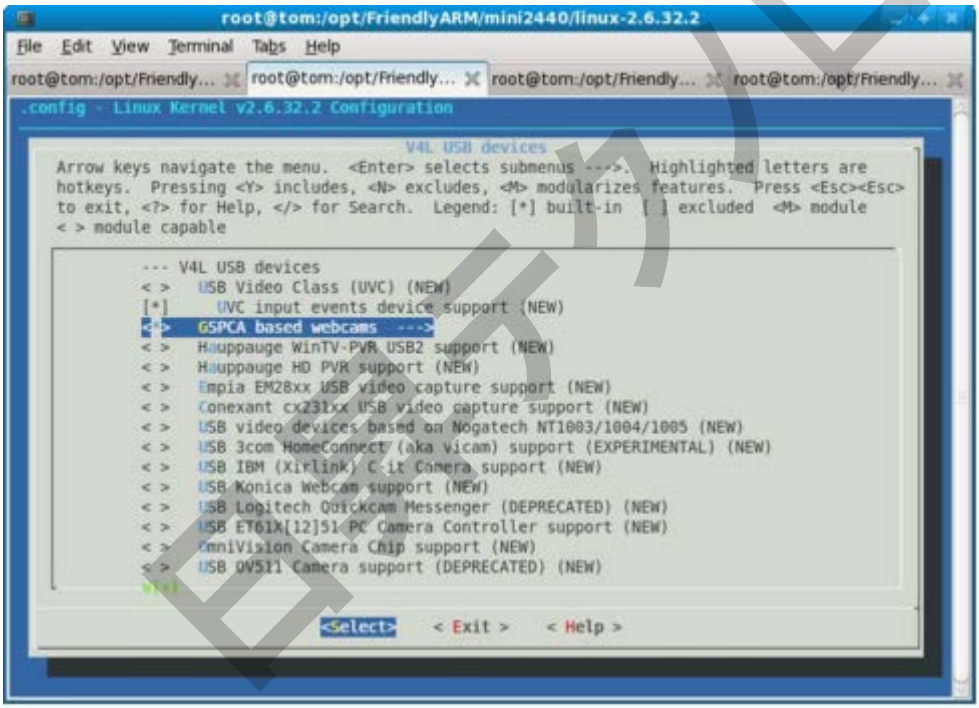
〝*〞オプションを選択し、Video capture adapters を選択し、入る



メニューでオプションを探し、入る



GSPCA based webcams を選択し、入る



GSPCA はユニバーサルタイプの USB カメラドライバプログラムであり、サポート項目をすべて選択、下図を参照する。

```
--- GSPCA based webcams
<*> ALi USB m5602 Camera Driver
<*> STV06XX USB Camera Driver
<*> GL860 USB Camera Driver
<*> Conexant Camera Driver
<*> Etoms USB Camera Driver
<*> Fujifilm FinePix USB V4L2 driver
<*> Jeilin JPEG USB V4L2 driver
<*> Mars USB Camera Driver
<*> Mars-Semi MR97310A USB Camera Driver
<*> OV519 USB Camera Driver
<*> OV534 USB Camera Driver
<*> Pixart PAC207 USB Camera Driver
<*> Pixart PAC7311 USB Camera Driver
<*> SN9C20X USB Camera Driver
<*> SONIX Bayer USB Camera Driver
<*> SONIX JPEG USB Camera Driver
<*> SPCA500 USB Camera Driver
<*> SPCA501 USB Camera Driver
<*> SPCA505 USB Camera Driver
<*> SPCA506 USB Camera Driver
<*> SPCA508 USB Camera Driver
<*> SPCA561 USB Camera Driver
<*> SQ Technologies SQ905 based USB Camera Driver
<*> SQ Technologies SQ905C based USB Camera Driver
<*> Syntek DV4000 (STK014) USB Camera Driver
<*> SUNPLUS USB Camera Driver
<*> T613 (JPEG Compliance) USB Camera Driver
<*> TV8532 USB Camera Driver
<*> VC032X USB Camera Driver
<*> ZC3XX USB Camera Driver
```


ここまでユニバーサルタイプの USB カメラドライバを設定した、新しいカーネルバージョンの場合、項目も追加する

3.16.6 USB カメラをテスト

カーネルソースコードディレクトリに make zImage を実行し、生成されたカーネルを開発ボードにプログラミングし、デフォルト・ファイルシステム root_qtopia を使用し、USB カメラダイナミックプレビューと

撮影プログラムがある

カメラを開発ボードの USB Host ポートと接続、｀USB カメラ｀プログラムをクリックし、撮影画面に入りフォーカス、パラメーターを設定出来る。｀Snap｀で撮影する。撮った写真はドキュメントグループに保存される。

備考：

良くある質問：USB カメラは Web ブラウザブラウザで見れません

mjpeg ソフトウェアを使用し、ネットを通じて USB カメラを制御、ブラウザでアクセスする時、次の情報が出る

```
[root@FriendlyARM /mjpg-streamer]# ./start_uvc_yuv.sh
MJPG Streamer Version.: 2.0
 i: Using V4L2 device.: /dev/video0
 i: Desired Resolution: 640 x 480
 i: Frames Per Second.: 5
 i: Format.....: YUV
 i: JPEG Quality.....: 80
Pixel format is unavailable, using JPEG
should never arrive exit fatal !!
 i: init_VideoIn failed
```

原因：mjpeg ソフトウェアは USB カメラタイプをサポートしない

解決方法：

(1) 別タイプの USB カメラに変更

(2) USB カメラドライバプログラムインタフェースをサポートするように、mjpeg のソースコードを変更する

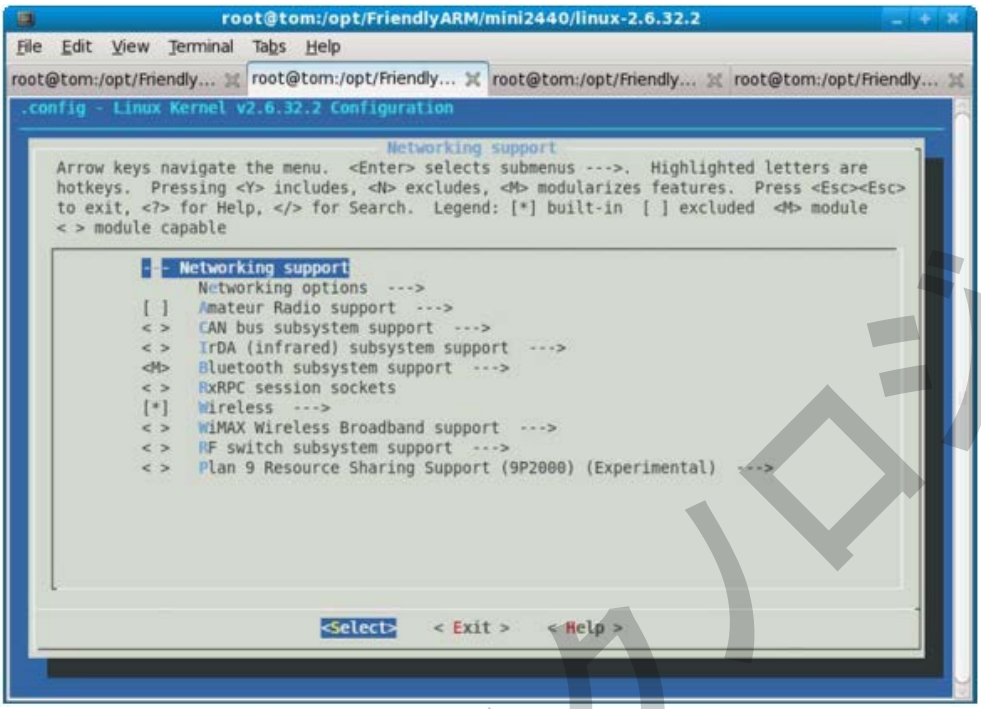
説明：Linux-2.6.32 などのハイバージョンカーネルはユニバーサルタイプの USB カメラドライバをサポートするが、ドライバ毎に上層に提供する画像ディコードインターフェースは異なるため、mjpeg ソフトウェアは一部分モデルの USB カメラしか認識できない。提供する Qtopia バージョンの US B カメラダイナミックプレビュープログラムはユーザーのフィード・バックより、より多い USB カメラが上層に提供する画像ディ

コードインターフェースサポートを追加する。

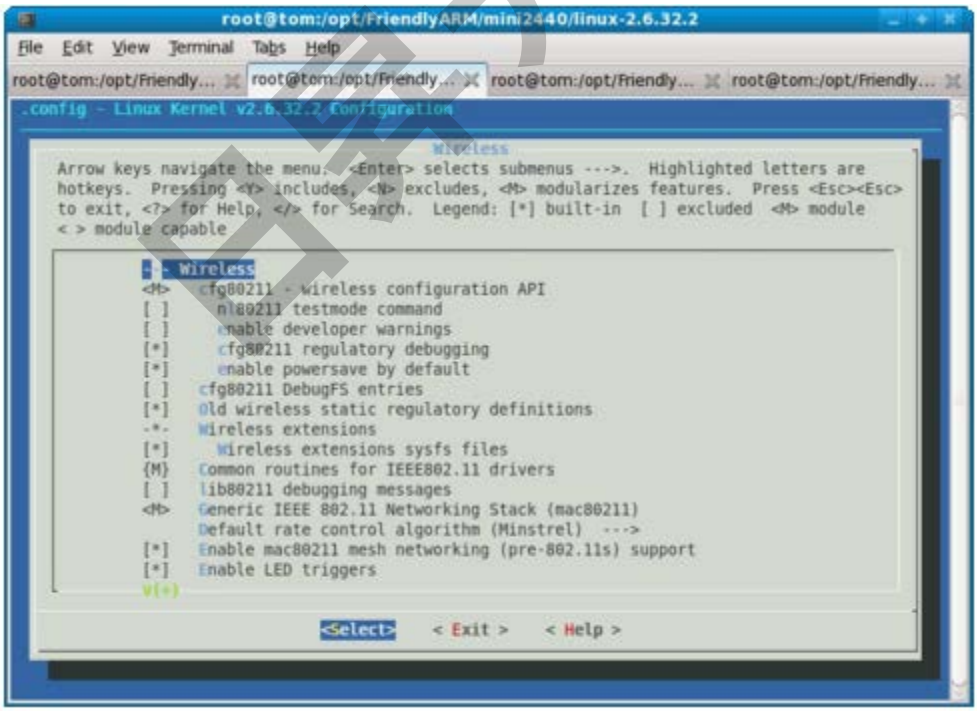
3.16.7 USB 無線 LAN の設定とテスト

新しい Linux-2.6.32.2 カーネルは多数 USB カメラドライバをサポートし、他に USB 無線 LAN もサポートし、次は USB 無線 LAN を追加する手順である

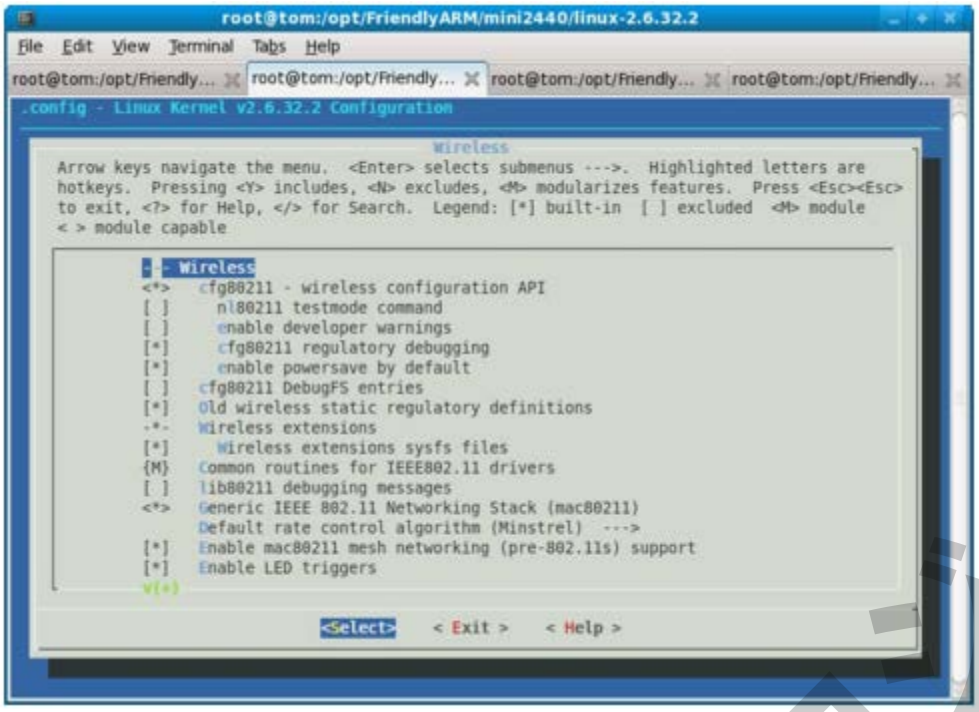
まずワイヤレスネットワークプロトコルを追加し、メインメニューに Netwoking support を選択し、Enter で入る



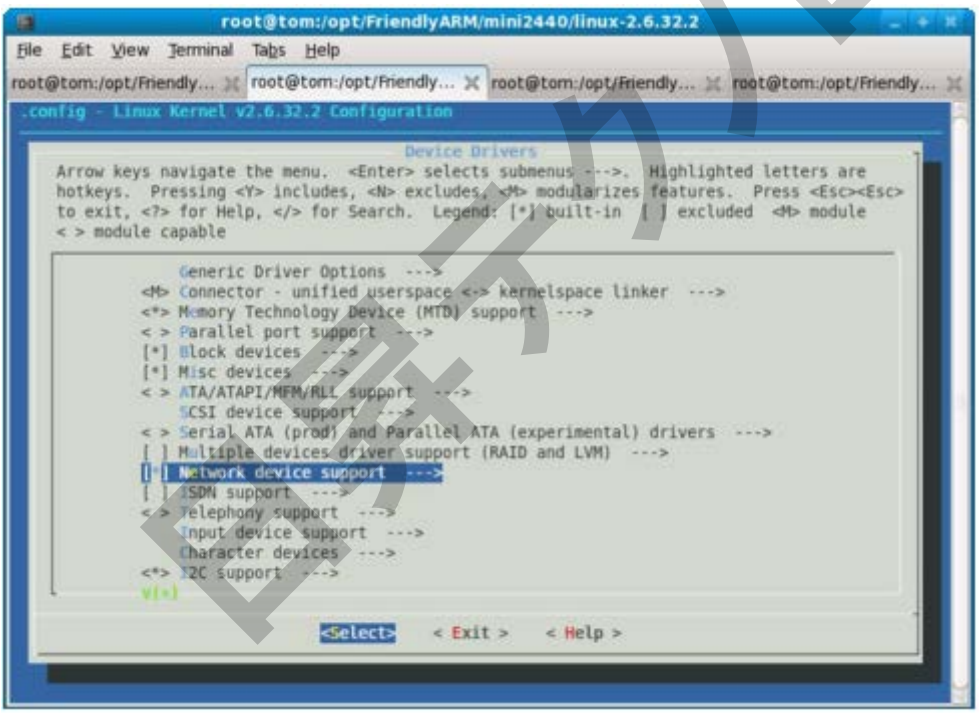
图のようなサブメニューが出て、Wireless を選択し、入る。ワイヤレスネットワークプロトコルを設定始める



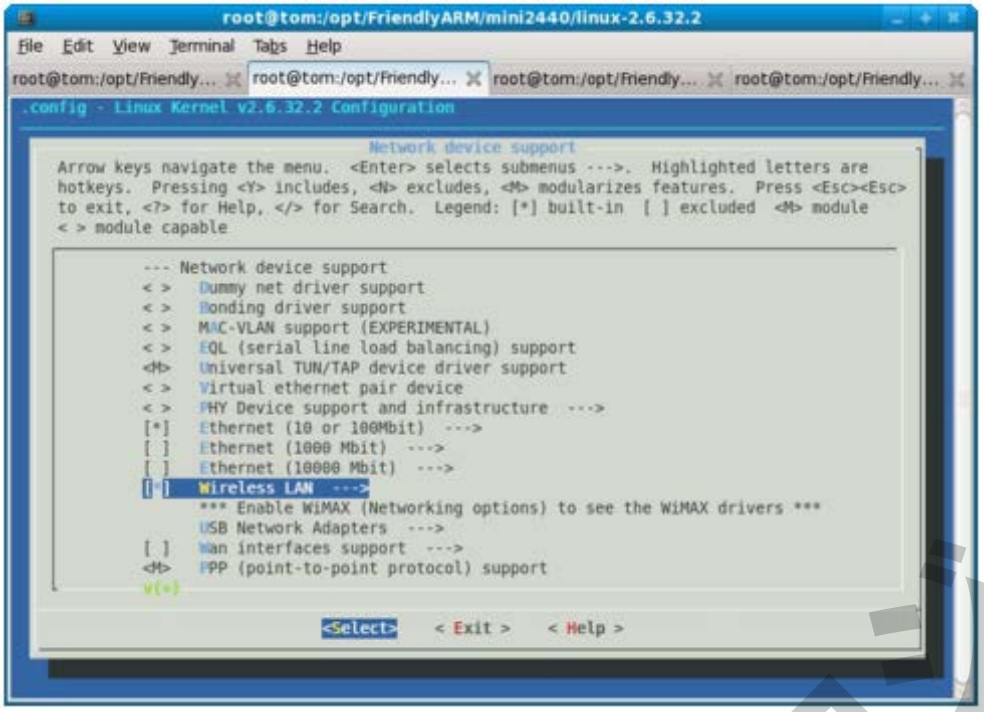
〝 *〝 の各項設定を選択する



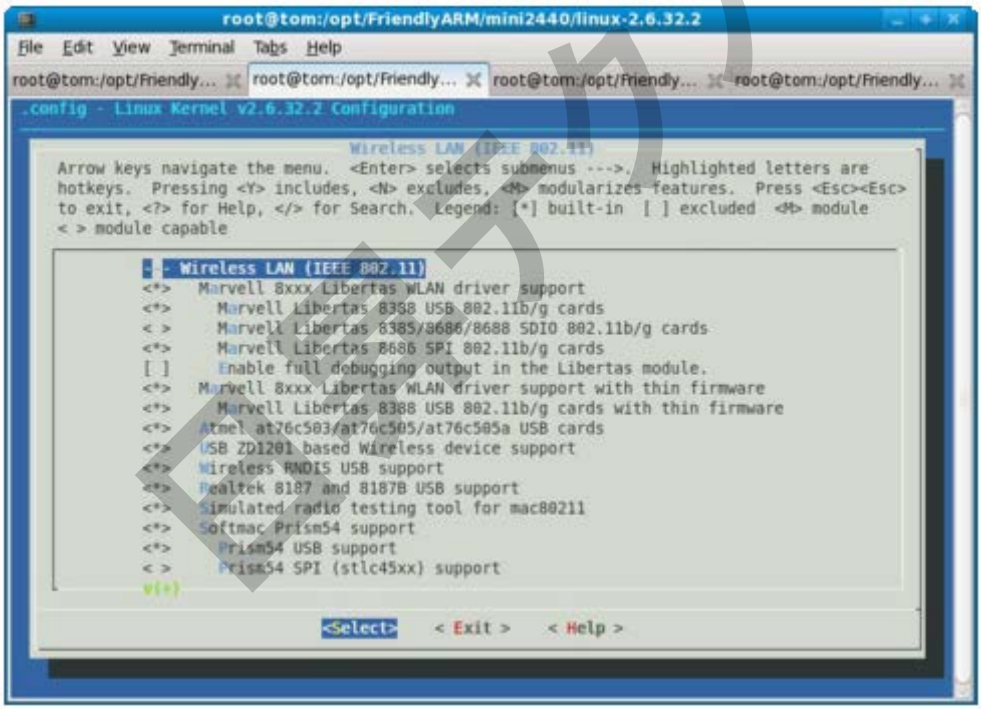
カーネル設定メインメニューに戻り、Device Drivers を選択して入る。ワイヤレスネットワークカードのドライバを設定する、下図を参照する



ネットワークデバイスサブメニューに入り、図のようにワイヤレスネットワークデバイスサブオプションを探し、入る



また`[*] Wireless LAN (IEEE 802.11) ---> ` サブメニューを選択して入り、各モデル USB 無線 LAN 設定を選択する。実際の情況よりテストを選択し、付属 DVD は Linux-2.6.32.2 カーネルを移植する。全て選択。



3.16.8 USB 無線 LAN をテスト

3.16.9 USB シリアル変換を設定

3.16.10 USB シリアル変換テスト

3.17 SD カードドライバを移植

3.17.1 カーネルに SD デバイスドライバを登録

Linux-2.6.32.2 は既に S3C2440 チップの SD カードドライバがあり、初期化コードに SD プラットフォームデバイス構造を追加するだけ、arch/arm/mach-s3c2440/mach-mini2440.c をオープンし、nand flash プラットフォーム構造に、次の赤いコードを追加する。

```
; mini2440.c top に SD カードデバイス構造の必要なヘッダファイルを追加します
#include <linux/mmc/host.h>
#include <plat/mci.h>

static struct platform_device mini2440_device_eth = {
        .name           = "dm9000",
        .id             = -1,
        .num_resources    = ARRAY_SIZE(mini2440_dm9k_resource),
        .resource       = mini2440_dm9k_resource,
        .dev            = {
        .platform_data    = &mini2440_dm9k_pdata,
        },
};
/* MMC/SD */

static struct s3c24xx_mci_pdata mini2440_mmc_cfg = {
.gpio_detect = S3C2410_GPG(8),
.gpio_wprotect = S3C2410_GPH(8),
.set_power = NULL,
.ocr_avail = MMC_VDD_32_33|MMC_VDD_33_34,
};
```

SD カード構造デバイスをターゲットプラットフォームデバイスセットに追加され、下図を参照する

```
static struct platform_device *mini2440_devices[] __initdata = {
        &s3c_device_usb,
        &s3c_device_rtc,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c0,
        &s3c_device_iis,
        &mini2440_device_eth,
        &s3c24xx_uda134x,
        &s3c_device_nand,
        &s3c_device_sdi,
};
```

SD カードのドライバプログラム最下層操作は実際にソースコード linux-2.6.32.2/drivers/mmc/host/s3cmci.c を対応し、テストによると、カーネルプリントアウト情報を含

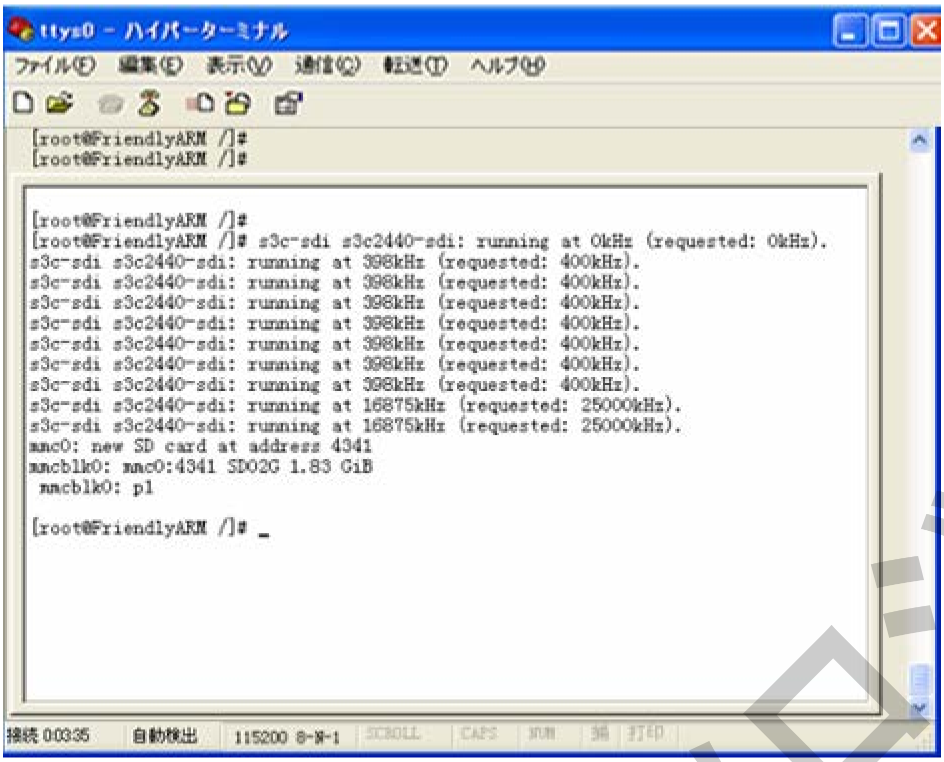
```
;遅延関数のヘッダファイル
#include <linux/delay.h>
static void pio_tasklet(unsigned long data)
{
        struct s3cmci_host *host = (struct s3cmci_host *) data;

        s3cmci_disable_irq(host, true);
        udelay(50); //ここで遅延関数を加えます

        if (host->pio_active == XFER_WRITE)
                do_pio_write(host);
```

まれる場合、SD カードは正常に認識されて使える。プリントアウト情報がない場合、不安定になり、該当プログラムに遅延コードを追加する、上図を参照する

SD カードの移植は完了。

3.17.2 SD カードをテスト

上記の手順に続き、カーネルソースコードディレクトリに make zImage を実行し、生成されたカーネルは開発ボードにプログラミングし、まず、SD カードを差し込まず（差し込む時のプリントアウト情報を確認するため）、システムが起動してから、コマンドラインコンソールに入り、この時 SD カードを差し込むと、次の情報を見える



開発ボードの/sdcard ディレクトリは自動的にマウントされ、Qtopia システムにタスクバーでアイコンを確認できる、下図を参照する

SD カードまた USB メモリードライブにあるすべてファイルは文書グループにすべて表示され、しかし、ディレクトリ名前を表示しない、ファイルは多すぎると、リスト数はかなり多くとなる

説明：Qtopia 2.2.0 プラグインを通じて SD カードまた USB メモリードライブ自動マウントをサポートする、MMC/SD カードまたは USB メモリードライブの第一のパーティションとして認識され、またフォーマットは VFAT/FAT32/FAT16 で、USB メモリードライブや SD カードは認識できない場合、VFAT/FAT32/FAT16 フォーマットを確認ください

3.17.3　mini2440 の SD カードドライバ分析

参考資料

1.SD Memory Card Specifications / Part 1. Physical Layer Specification; Version 1.0

2.LDD3 CHAPTER-16 BLOCK DEVICE

3. http://www.sdcard.org

1. ハードウェア基礎

SD（セキュリティデジタル）と MMC（マルチメディアカード）

SD は、一般に、共通の SD メモリカードであり、MMC は、メモリカード規格の以前の一種であり、現在 SD 規格によって置換されているフラッシュメモリカード規格である。

SDIO（セキュリティ･デジタル I / O）

SDIO は、SDIO の名前通り、SD I / O インタフェースの意味、

これは少し抽象的な解釈かもしれない。具体的に説明するように、SD メモリーカードの規格はもともとあったが、今では SD にいくつかのペリフェラル･インターフェースを挿入するために使用することができ、そのような技術が SDIO。

SDIO 自体は外部の周辺機器を接続するための SD I / O ピンを介して、非常に単純な技術であり、SD I / O を経由で外部と接続し、それはまた、SD 上の I/O が外部とデータを転送します。そして、SD アソシエーションのメンバーは完全な SDIO スタックドライバを出し、SDIO 周辺機器（我々は SDIO カードと呼ばれる）の開発と応用は非常に普及している。

そこに携帯電話やハンドヘルドデバイスからサポートする SDIO 機能（SD 標準は、もともとモバイル機器に対して開発されたもの）が多くなり、そして、数多くの SDIO 周辺機器も開発されている、携帯に外部周辺機器をより簡単に接続され、開発上にも拡張性がある。（内蔵ペリフェラル不要）。

現在の一般的な SDIO 周辺機器（SDIO カード）次のとおり。

Wi-Fi カード（無線ネットワークカード）
CMOS センサーカード（カメラモジュール）
GPS カード
GSM / GPRS モデムカード
Bluetooth カード
ラジオ/ TV カード

SDIO アプリケーションは、組込みシステムの将来になる最も重要なインターフェース技術の一つであり、現在の GPIO タイプ SPI インタフェースに置き換えられる。

SD / SDIO 転送モードは、次の三種類がある。

SPI モード（必須）
1 ビットモード
4 ビットモード

SD の MMC モード

SD はまた MMC メモリを読むことができ、MMC 規格に述べた通り、MMC メモリは必ずしも SPI モード（ただし、1 ビットモードをサポートするようにしてください）をサポートしたいのですが、市販の MMC カードは殆ど SPI モードをサポートする。そこで、MMC メモリカードを読み取るために SD カードを SPI モードに設定する。

SD の MMC モードが MMC カードを読み取るために使用されるもの、SD の SPI モードは MMC モードを使用されるが、物理特性は異なる。

MMC SPI モードの最大伝送速度：20 メガビット/秒

SD SPI モードの最大伝送速度：25 メガビット/秒

混乱を避けるため、SPI/MMC モードと SPI/SD モードを分けて書く。

2. MMC サプシステムの基本フレームワーク仕組み

残念ながら、カーネルは私たちに MMC サプシステムに関するドキュメントを提供されなく、google に検索しても、関連の文章を見つけないから、自分でコード分析を勉強する。MMC サプシステムのコードは kernel/driver/MMC のもとにあり、現在の MMC サプシステムはあるフォーマットのメモリカード：SD、SDIO、MMC をサポートします。MMC サプシステムは三つ部分に分けられる

HOST 部分は異なるホストに対するプログラムで、ドライバプログラムエンジニアがプラットフォームの特性よりこの部分を完成する

CORE 部分は全体 MMC のコアメモリで、異なるプロトコルと規格の実現を完成し、HOST 層のドライバにインタフェース関数を提供する

CARD 部分：これらのメモリカードは、ブロック·デバイスであるから、SD カードはどのようにブロック·デバイスになるかを実現する

3.HOST 層分析

HOST 層は特定のホストに対するドライバプログラムを実現し、ここで、mini2440 の s3cmci.c を例として分析する。platform_driver_register(&s3cmci_2440_driver)を使用し、プラットフォームデバイスを登録し、次に probe 関数をメイン説明する。この関数に、CORE との関係は次の三つことを通じて実現する、まず、mmc_host 構造を割り当てられ (sizeof(struct s3cmci_host)にご注意)、このように mmc_host から s3cmci_host を見つけられる、組み込みと組み込まれる構造は相手を見つけると言う技術は Linux カーネルによく使われる。これから、MMC-> pos に値を与える、s3cmci_ops 構造はいくつかの重要な関数を実現し、後は説明する、mmc 構造の多くのメンバーに値を与える、最後に mmc 構造は MMC サブシステムに追加し、mmc_alloc_host と mmc_add_host の具体てきな役割は次の章節に説明する、この三つのコマンドは MMC ドライバ層を書くには必要である

```
mmc = mmc_alloc_host(sizeof(struct s3cmci_host)、 &pdev->dev);
mmc->ops     = &s3cmci_ops;
.....
```

s3cmci_ops に四つ関数を含める：

```
static struct mmc_host_ops s3cmci_ops = {
.request    = s3cmci_request、
.set_ios    = s3cmci_set_ios、
.get_ro       = s3cmci_get_ro、
.get_cd       = s3cmci_card_present、
};
```

簡単のものから分析し、これらの関数は core 部分に呼び出される

s3cmci_get_ro:この関数は GPIO から読み取ってカードは保護されるかどうか判断する

s3cmci_card_present：この関数は GPIO から読み取ってカードが存在するかどうか判断する

s3cmci_set_ios：s3cmci_set_ios(struct mmc_host *mmc、 struct mmc_ios *ios)

コア層から受信した ios よりハードウェアの IO を設定し、ピン・コンフィグ、クロック・有効、バス帯域幅設定などを含まれる。

s3cmci_request：この関数は最も重要で複雑な関数で、コマンドとデータの送信と受信を実現し、CORE 部分はコマンドやデータを送信する時、この関数を呼び出し、mrq リクエストを伝送する

```
static void s3cmci_request(struct mmc_host *mmc、 struct mmc_request *mrq)
{
struct s3cmci_host *host = mmc_priv(mmc);
host->status = ″mmc request″;
host->cmd_is_stop = 0;
host->mrq = mrq;

if (s3cmci_card_present(mmc) == 0) {
dbg(host、 dbg_err、 ″%s: no medium present¥n″、 __func__);
host->mrq->cmd->error = -ENOMEDIUM;
mmc_request_done(mmc、 mrq);//カードがない場合、リクエストを終了する
} else
s3cmci_send_request(mmc);
}
```

次には s3cmci_send_request(mmc)を見て：

この関数はリクエストの時、データを伝送するか、またはコマンドを伝送するかと判断し、データの場合

s3cmci_setup_data を呼び出し、S3C2410_SDIDCON レジスタを設定し、その後、SDITIMER レジスタを設定し、このようにバス幅を設定した、DMA を使用するかどうか、データ伝送モードを起動するかどうか、次の割り込みを有効になる：

```
imsk = S3C2410_SDIIMSK_FIFOFAIL | S3C2410_SDIIMSK_DATACRC |
S3C2410_SDIIMSK_DATATIMEOUT | S3C2410_SDIIMSK_DATAFINISH;
```

DMA でデータを送信するか、または FIFO でデータを送信するかと判断する

```
if (host->dodma)
/    because host->dodma         = 0、so we don't use it
res = s3cmci_prepare_dma(host、 cmd->data);//DMA 送信を準備する、
else
res = s3cmci_prepare_pio(host、 cmd->data);.//FIFO 送信を準備する
```

コマンドの場合:s3cmci_send_command () というコマンド送信関数を呼び出し、datesheet に説明するプロセスは大体同じで、SD 規格のコマンドフォーマットについて参考資料 1 を

参照する

```
  writel(cmd->arg、 host->base + S3C2410_SDICMDARG);/*先にパラメータレジスタを書き込む
  ccon  = cmd->opcode & S3C2410_SDICMDCON_INDEX;//コマンド種類を設定
  ccon |= S3C2410_SDICMDCON_SENDERHOST | S3C2410_SDICMDCON_CMDSTART;
  /*with start 2bits*/
  if (cmd->flags & MMC_RSP_PRESENT)
  ccon |= S3C2410_SDICMDCON_WAITRSP;
  /*wait rsp*/
  if (cmd->flags & MMC_RSP_136)
  ccon |= S3C2410_SDICMDCON_LONGRSP;
  //respose のタイプを確認
  writel(ccon、 host->base + S3C2410_SDICMDCON);
```

コマンドチャンネルを分析してから、データチャンネルを分析し、まずFIFO方式でどのように伝送を実現するかを分析する

```
  s3cmci_prepare_pio(host、 cmd->data)を分析し
  rw より読み取り/書き込みを判断し
  if (rw) {
  do_pio_write(host);
  /* Determines SDI generate an interrupt if Tx FIFO fills half*/
  enable_imask(host、 S3C2410_SDIIMSK_TXFIFOHALF);
  } else {
  enable_imask(host、 S3C2410_SDIIMSK_RXFIFOHALF
  | S3C2410_SDIIMSK_RXFIFOLAST);
  }
```

データはSDに書き込まれる場合、do_pio_write を呼び出し、FIFOにデータを充填し、64 ビットのFIFOは33ビットより少ない場合、割り込みが生じる、SDからデータを読み取る場合、まず割り込みを有効にする、FIFOは31ビットより多くになる場合、割り込みサービスプログラムを呼び出し、割り込みサービスプログラムはdo_pio_read FIFOのデータを呼び出し、読み取る

```
  次に do_pio_write を分析し
  to_ptr = host->base + host->sdidata;
  fifo_free(host)は fifo の残り容量を検出するために使用される
  while ((fifo = fifo_free(host)) > 3) {
  if (!host->pio_bytes) {
  res = get_data_buffer(host、 &host->pio_bytes、
  /* If we have reached the end of the block、 we have to
  * write exactly the remaining number of bytes.  If we
```

```
  * in the middle of the block、 we have to write full
  * words、 so round down to an even multiple of 4. */
  if (fifo >= host->pio_bytes)//fifo の容量は pio_bytes より大きい場合、最後にこのブロックを読み取ることを明らかにする
  fifo = host->pio_bytes;
  /* because the volume of FIFO can contain the remaning block*/
  else
  fifo -= fifo & 3;/*round down to an even multiple of 4*/

  host->pio_bytes -= fifo;//余った文字を更新する
  host->pio_count += fifo;/*chang the value of pio_bytes*/

  fifo = (fifo + 3) >> 2;//ビット数は文字数に変換される
  /*how many words fifo contain、every time we just writ one word*/
  ptr = host->pio_ptr;
  while (fifo--)
  writel(*ptr++、 to_ptr);// FIFO へ書き込む.
  host->pio_ptr = ptr;
  }
```

注釈一：MMC コアは mrq->data メンバーに struct scatterlist 表を割り当てられ、スキャッター･ギャザーをサポートするために使用され、この方法を通じて、物理における一致しないメモリページは連続的な配列に組み立てられる、大きなバッファを割り当てることを避ける

コードを見て

```
  if (host->pio_sgptr >= host->mrq->data->sg_len) {
  dbg(host、 dbg_debug、 "no more buffers (%i/%i)¥n"、
  host->pio_sgptr、 host->mrq->data->sg_len);
  return -EBUSY;
  }
  sg = &host->mrq->data->sg[host->pio_sgptr];
      *bytes = sg->length;// ページバッファの長さ
  *pointer = sg_virt(sg); ページアドレスは仮想アドレスにマッピングされる
  host->pio_sgptr++;ここで、プログラムは一回マッピングを完了することを現れる
```

このように mmc リクエスト毎に対して scatterlist 表の一つページ（ブロック）を処理するだけ、フルリクエストを完了するのは sg_len をマッピングする必要がある

Mmc 書きデバイスのリクエストプロセスをまとめて

s3cmci_prepare_pio の中、始めに do_pio_write を呼び出し、FIFO 空間は 3 より大きくなって、また、scatterlist を取得できる場合、FIFO へデータを書き始め、FIFO 容量は 3 より小さくなる場合、TXFIFOHALF 割り込み・オンする、サービスプログラムを割り込みす

る時、TFDET を検出すると、FIFO でスペースがあると表示し、この時、TXFIFOHALF 割り込みをオフし、do_pio_write を呼び出し、書き込む

データの流れは次の通りであり：scatterlist-------->fifo---------->sdcard

Mmc 読み取りデバイスのリクエストのプロセスデータの流れは次の通りであり：sdcard --------> fifo ---------->scatterlist、データの読み取りプロセスと割り込みのトリガについて、s3cmci_prepare_pio の中の enable_imask(host、

S3C2410_SDIIMSK_RXFIFOHALF、S3C2410_SDIIMSK_RXFIFOLAST)；SD カードからデータを読み取らない場合、割り込みを引き起こす、S3C2410_SDIIMSK_RXFIFOLAST から引き起こすプロセスは

次に割り込みサービスプログラムを分析し

static irqreturn_t s3cmci_irq(int irq、 void *dev_id)

該当プログラムはすべてステータスレジスタを取得する

mci_csta = readl(host->base + S3C2410_SDICMDSTAT);

mci_dsta = readl(host->base + S3C2410_SDIDSTA);

mci_dcnt = readl(host->base + S3C2410_SDIDCNT);

mci_fsta = readl(host->base + S3C2410_SDIFSTA);

mci_imsk = readl(host->base + host->sdiimsk);

これらは割り込み処理の根拠とする

DMA モードではない場合、データの受送信を処理する

```
if (!host->dodma) {
if ((host->pio_active == XFER_WRITE) &&
(mci_fsta & S3C2410_SDIFSTA_TFDET)) {
/*This bit indicates that FIFO data is available for transmit when
DatMode is data transmit mode. If DMA mode is enable、 sd
host requests DMA operation.*/
disable_imask(host、 S3C2410_SDIIMSK_TXFIFOHALF);
tasklet_schedule(&host->pio_tasklet);
```

注： tasklet の遅延機能を使用し、割り込みサービス時間を短縮する、遅延関数 pio_tasklet に do_pio_write と do_pio_read を呼び出す

```
host->status = "pio tx";
}

if ((host->pio_active == XFER_READ) &&
(mci_fsta & S3C2410_SDIFSTA_RFDET)) {
disable_imask(host、
S3C2410_SDIIMSK_RXFIFOHALF |
S3C2410_SDIIMSK_RXFIFOLAST);

tasklet_schedule(&host->pio_tasklet);
```

```
host->status = "pio rx";
}
```

次の多く k のコードは他のタイプの割り込みに対する処理である

最後に DMA モードを分析し、このモードで CPU の介入が必要ではない、S3C2440 の DMA は四つチャンネルがあり、チャンネル 0 を選択する

```
static int s3cmci_prepare_dma(struct s3cmci_host *host、 struct mmc_data *data)
{
int dma_len、 i;
int rw = (data->flags & MMC_DATA_WRITE) ? 1 : 0;

BUG_ON((data->flags & BOTH_DIR) == BOTH_DIR);

s3cmci_dma_setup(host、 rw ? S3C2410_DMASRC_MEM : S3C2410_DMASRC_HW);//注一
s3c2410_dma_ctrl(host->dma、 S3C2410_DMAOP_FLUSH);

dma_len = dma_map_sg(mmc_dev(host->mmc)、 data->sg、 data->sg_len、
(rw) ? DMA_TO_DEVICE : DMA_FROM_DEVICE);//注二

if (dma_len == 0)
return -ENOMEM;

host->dma_complete = 0;
host->dmatogo = dma_len;

for (i = 0; i < dma_len; i++) {
int res;

dbg(host、 dbg_dma、 "enqueue %i:%u@%u¥n"、 i、
sg_dma_address(&data->sg)、
sg_dma_len(&data->sg));

res = s3c2410_dma_enqueue(host->dma、 (void *) host、
sg_dma_address(&data->sg)、
sg_dma_len(&data->sg));

if (res) {
s3c2410_dma_ctrl(host->dma、 S3C2410_DMAOP_FLUSH);
return -EBUSY;
}
```

```
}

s3c2410_dma_ctrl(host->dma、 S3C2410_DMAOP_START);

return 0;
}
```

注一:この関数はまず s3c2410_dma_devconfig を呼び出し、DMA 送信元/ターゲットのタイプとアドレスを設定する。メモ：ここのデバイス host->mem->start + host->sdidata は実は SDIDATA レジスタのアドレス値であり、SD カードを書き込むと、ターゲットアドレスであり、ではなければ、ソースアドレスである、これから s3c2410_dma_set_buffdone_fn(host->dma、 s3cmci_dma_done_callback)を呼び出す

DMA チャンネル 0 のコールバック関数を設定する

注二：

```
dma_len = dma_map_sg(mmc_dev(host->mmc)、 data->sg、 data->sg_len、
(rw) ? DMA_TO_DEVICE : DMA_FROM_DEVICE);
```

ここで、スキャッタ/ギャザーマッピング（P444、LDD3）を行い、戻り値は伝送される DMA バッファの数であり、sg_len より小さいな場合はある、即ち sg_len と dma_len は違うのである

```
sg_dma_address(&data->sg)、戻り値はバス（DMA）アドレス
sg_dma_len(&data->sg))；戻り値はバッファの長さである。
```

最後に呼び出すのは s3c2410_dma_enqueue(host->dma、 (void *) host、

```
sg_dma_address(&data->sg)、
sg_dma_len(&data->sg))；
```

各 DMA バッファに行列に并んで、処理を待つ

```
s3c2410_dma_ctrl(host->dma、 S3C2410_DMAOP_START)；DMA を起動する
```

これによって DMA バッファは scatterlist と繋がる。データを書き込む時、scatterlist のデータは上記のマッピング関係のため、DMA バッファに直接にコピーされる、データを読み取る時、これとは逆に。すべて処理は CPU の介入が必要ではない、自動的に完成する

4. CORE 層分析

CORE 層は異なるプロトコルと規格を実現させ、そして、HOST 層のドライバにインタフェース関数を提供する、HOST 層に呼び出された二つ関数は次の通り

```
mmc_alloc_host(sizeof(struct s3cmci_host)、 &pdev->dev);
mmc_add_host(mmc);
```

この二つ関数から始め、CORE 層と HOST 層はどのようにお互いににやり取りすることを分析する

```
まず、mmc_alloc_host 関数を見て
dev_set_name(&host->class_dev、 "mmc%d"、 host->index);
host->parent = dev;
```

host->class_dev.parent = dev;
host->class_dev.class = &mmc_host_class;
device_initialize(&host->class_dev);
上記関数は/SYS/CLASS/mmc_host 下に mmc0 ディレクトリを生成する、クラス・デバイスを追加し、2.6.21 以降のバージョンにクラス・デバイスの class_device は device に取り替えられ、LDD3P387 の内容は古い。
INIT_DELAYED_WORK(&host->detect、 mmc_rescan);
動作キューを初期化、遅延関数は mmc_rescan で、最後に host に対してデフォルト設定を行う、probe 関数の後で再設定する
mmc_add_host(mmc)を分析する;
device_add(&host->class_dev);クラス・デバイスを追加する。
その中で mmc_start_host を呼び出す
void mmc_start_host(struct mmc_host *host)
{
mmc_power_off(host);
mmc_detect_change(host、 0);
}
mmc_power_off に ios に対して設定を行い、その後 mmc_set_ios(host)を呼び出す;
host->ios.power_mode = MMC_POWER_OFF;
host->ios.bus_width = MMC_BUS_WIDTH_1;
host->ios.timing = MMC_TIMING_LEGACY;
mmc_set_ios(host);
mmc_set_ios(host)のキー関数 host->ops->set_ios(host、 ios);set_ios は前の set_ios = s3cmci_set_ios である
次に mmc_detect_change(host、 0)を見て;最後関数
mmc_schedule_delayed_work(&host->detect、 delay)は;
前の遅延関数 mmc_rescan を呼び出す
mmc_power_up(host);//この関数は前の mmc_power_off と類似、起動時必要な ios を設定する
mmc_go_idle(host);
//CMD0 、from inactive to idle
mmc_send_if_cond(host、 host->ocr_avail);// SD_SEND_IF_COND を送信し、SD2.0 カードを使用する時、設定コマンド
/*suppot for 2.0 card*/
* ...then normal SD...
*/
err = mmc_send_app_op_cond(host、 0、 &ocr);
if (!err) {
if (mmc_attach_sd(host、 ocr))

```
mmc_power_off(host);
goto out;
}
```

上記は、SD カードプロトコルの SD カード起動流れに従い、非アクティブモード、カード認識モードとデータ伝送モードの三つモードで合計 9 つの状態変換を含む。関連規格を参照し、次の三章図を参照し、モードと状態、状態変換を理解する

SD カードの初期状態は inactive 状態で mmc_go_idle(host)を呼び出してからコマンド CMD0 を送信し、IDLE 状態にさせるため

mmc_go_idle を分析し

```
memset(&cmd、 0、 sizeof(struct mmc_command));
cmd.opcode = MMC_GO_IDLE_STATE; //MMC_GO_IDLE_STATE は CMD0 コマンドである
cmd.arg = 0;//このコマンドはパラメータがない
cmd.flags = MMC_RSP_SPI_R1 | MMC_RSP_NONE | MMC_CMD_BC;
err = mmc_wait_for_cmd(host、 &cmd、 0);//注 1 を参照
mmc_delay(1);
```

```
注 1：mmc_wait_for_cmd(host、 &cmd、 0)はコマンド送信用。
memset(&mrq、 0、 sizeof(struct mmc_request));
memset(cmd->resp、 0、 sizeof(cmd->resp));
cmd->retries = retries;
mrq.cmd = cmd;コマンドは mmc リクエストに組み込まれる
cmd->data = NULL;mmc コマンドの data 部分は NULL に設定され、伝送するのはコマンドであり、データではないことを現れる
mmc_wait_for_req(host、 &mrq);// 肝心な部分
```

該当関数には mmc_start_request を呼び出し、この関数は host->ops->request(host、mrq)を呼び出し、この request 関数はこの前の部分に分析する s3cmci_request である。次のことを見て

```
err = mmc_send_app_op_cond(host、 0、 &ocr);//注一
if (!err) {
if (mmc_attach_sd(host、 ocr))//注二
mmc_power_off(host);
goto out;
```

注一：実際には ACMD41 コマンドを送信し、このコマンドは SDcard の許可する電圧範囲値を取得するに使用され、これは応用コマンドのため、送信する前に CMD_55 コマンドを送信する必要がり、実行してから card 状態は READY が取得した電圧範囲値になって、ocr に保存される、mmc_attach_sd(host、 ocr)を呼び出し、この電圧範囲はホストの要求を確認し、満たすことできない場合、power_off ホストである

注二：mmc_attach_sd はマッチ、カード初期化機能を実現する

host->ocr = mmc_select_voltage(host、 ocr);マッチかどうかを確認し、マッチの場合、次の初期化操作を行い

mmc_sd_init_card(host、 host->ocr、 NULL);この関数を分析し

(1) mmc_all_send_cid () この関数は CMD2 を発生し、カードの ID 情報を取得し、ID 状態に入る

(2)card = mmc_alloc_card(host、 &sd_type);一枚 SD タイプの card 構造を割り当てられる

(3)mmc_send_relative_add を呼び出し、カード対応アドレスを取得し、注：前のカードとホスト通信はデフォルトアドレスを使用し、stand_by 状態に入る

(4) SEND_CSD (CMD9)の送信を通じて CSD レジスタの情報を取得し、 block 長さ、カード容量を含まれる

(5) mmc_select_card(card)は CMD7 を送信し、現在の RADD アドレスのカードを選択し、どんな時にバスにただ一枚カードは選択される、伝送状態に入る、

(6) mmc_app_send_scr を呼び出し、コマンド ACMD51 を送信し、SRC レジスタの内容を取得し、SENDING-DATA 状態に入る

関数に取得した各カードレジスタの内容をディコードし、cmd 構造の該当のメンバーに保存される。

(7) if (host->ops->get_ro(host) > 0)

mmc_card_set_readonly(card);

get_ro(host)関数を呼び出すのを通じて、実際には s3cmci_get_ro 関数である、書き込み保護を判断し、card 状態は読み取り専用ステータス状態を設定される

最後に mmc_attach_sd に card 構造を追加する

mmc_add_card(host->card);

dev_set_name(&card->dev、 "%s:%04x"、 mmc_hostname(card->host)、 card->rca);ここで、host 名前+rca アドレスで、カードを名づける、

/sys/devices/platform/s3c2440-sdi/mmc_host:mmc0/の中に mmc0：0002 ディレクトリ、0002 は rca アドレスである

5. CARD 層分析

これらのメモリカードは、ブロック·デバイスであるため、ブロック·デバイスのドライバプログラムを提供するで、この部分は SD カードはどのようにブロック·デバイスを実現するかのもので、まず、block.C の probe 関数を見て

MMC ブロック·デバイスは次の構造で表示される

```
struct mmc_blk_data {
spinlock_t    lock;
struct gendisk    *disk;
struct mmc_queue queue;
unsigned int    usage;
unsigned int    read_only;
```

```
};
先まず mmc_blk_alloc( )を見て
devidx = find_first_zero_bit(dev_use、 MMC_NUM_MINORS);
if (devidx >= MMC_NUM_MINORS)// mmc 層は一番多くの 16 個カードをサポートするを現れる、card 毎には 8 パーティションを占める
return ERR_PTR(-ENOSPC);
__set_bit(devidx、 dev_use);

md->disk = alloc_disk(1 << MMC_SHIFT);// 一つディスク、8 つパーティションを割り当てる
//8 partion
if (md->disk == NULL) {
ret = -ENOMEM;
goto err_kfree;
}

spin_lock_init(&md->lock);
md->usage = 1;

ret = mmc_init_queue(&md->queue、 card、 &md->lock);//注一
if (ret)
goto err_putdisk;
md->queue.issue_fn = mmc_blk_issue_rq;//
md->queue.data = md;
md->disk->major    = MMC_BLOCK_MAJOR;
md->disk->first_minor = devidx << MMC_SHIFT;
md->disk->fops = &mmc_bdops; ディスクの操作関数
md->disk->private_data = md;
md->disk->queue = md->queue.queue;
md->disk->driverfs_dev = &card->dev;

/*
* As discussed on lkml、 GENHD_FL_REMOVABLE should:
*
* - be set for removable media with permanent block devices
* - be unset for removable block devices with permanent media
*
* Since MMC block devices clearly fall under the second
* case、 we do not set GENHD_FL_REMOVABLE.  Userspace
```

```
 * should use the block device creation/destruction hotplug
 * messages to tell when the card is present.
 */

 sprintf(md->disk->disk_name, "mmcblk%d", devidx);//名前は/proc/device の中に出る
 /sys/block の中に"mmcblk0 がある
 blk_queue_hardsect_size(md->queue.queue, 512);// ハードウェアセクタの容量を設定する

 }
 注一：
 mq->queue = blk_init_queue(mmc_request, lock);初期化、request 関数はキューとバインドされる
 if (!mq->queue)
 return -ENOMEM;
 mq->queue->queuedata = mq;
 mq->req = NULL;
 blk_queue_prep_rq(mq->queue, mmc_prep_request);
 //コマンドの前処理、ドライバプログラムは evl_next_request に戻る前に、チェックと前処理リクエストのメカニズムを提供し、LDD3 P485 を参照する
 //command prepare process
 blk_queue_ordered(mq->queue, QUEUE_ORDERED_DRAIN, NULL);//
 //barrier request バリアリクエスト､再度組み合わせることより発生するエラーを防ぐ､設定は標準に達してからリクエストデータはタイムリーにメディアに書き込まれることを保証する。
 mq->sg = kmalloc(sizeof(struct scatterlist) *
 host->max_phys_segs, GFP_KERNEL);
 if (!mq->sg) {
 ret = -ENOMEM;
 goto cleanup_queue;
 }
 sg_init_table(mq->sg, host->max_phys_segs);
 }
 //scatterlist 構造体を割り当てる
 mq->thread = kthread_run(mmc_queue_thread, mq, "mmcqd");最後にカーネルスレッドを設定し、スレッドの関連関数は mmc_queue_thread で、これは重要で後程分析する。
 これから mmc_blk_set_blksize を呼び出し、block の長さは 512 に設定される。
 すべて準備してからディスク add_disk(md->disk) をアクティベーションする;
```

最後に request 関数を分析する：

```
*
* Generic MMC request handler.  This is called for any queue on a
* particular host.  When the host is not busy、 we look for a request
* on any queue on this host、 and attempt to issue it.  This may
* not be the queue we were asked to process.即ち、elv_next_request から戻られた req は必ずしも mq->req ではない
*/
static void mmc_request(struct request_queue *q)
{
struct mmc_queue *mq = q->queuedata;
struct request *req;
int ret;
if (!mq) {
printk(KERN_ERR "MMC: killing requests for dead queue¥n");
while ((req = elv_next_request(q)) != NULL) {
do {
ret = __blk_end_request(req、 -EIO、
blk_rq_cur_bytes(req));//処理できるリクエストはない場合、このリクエストを使う
} while (ret);
}
return;
}
if (!mq->req)
wake_up_process(mq->thread);//注一
}
```

注一：LDD3 で紹介するブロックデバイス方法と異なって、bio 構造に関連するデバイスはない、リクエストが取得する時、データブロックの送信は mq->thread スレッドを呼び出すことにより実現できる、スレッドは実際には mmc_queue_thread 関数である

```
static int mmc_queue_thread(void *d)
{
struct mmc_queue *mq = d;
struct request_queue *q = mq->queue;

current->flags |= PF_MEMALLOC;

down(&mq->thread_sem);
do {
struct request *req = NULL;
```

```
spin_lock_irq(q->queue_lock);
set_current_state(TASK_INTERRUPTIBLE);
if (!blk_queue_plugged(q))
req = elv_next_request(q);
mq->req = req;
spin_unlock_irq(q->queue_lock);

if (!req) {
if (kthread_should_stop()) {
set_current_state(TASK_RUNNING);
break;
}
up(&mq->thread_sem);
schedule();
down(&mq->thread_sem);
continue;
}
set_current_state(TASK_RUNNING);
//
mq->issue_fn(mq、 req);//注一
} while (1);
up(&mq->thread_sem);

return 0;
}
注一:issue_fn 関数：
brq.data.sg = mq->sg;
brq.data.sg_len = mmc_queue_map_sg(mq);
/*
* Adjust the sg list so it is the same size as the
* request.
*/
if (brq.data.blocks != req->nr_sectors) {
int i、 data_size = brq.data.blocks << 9;
struct scatterlist *sg;

for_each_sg(brq.data.sg、 sg、 brq.data.sg_len、 i) {
data_size -= sg->length;
```

```
if (data_size <= 0) {
sg->length += data_size;
i++;
break;
}
}
brq.data.sg_len = i;
}
```

上記のコードは scatterlist を準備するため、データ伝送のバッファ

mmc_wait_for_req(card->host、 &brq.mrq);次に host へリクエストを送信し、この関数はよく分かるはずで、この最後は host->ops->request(host、 mrq)である、このようにドライバプログラムの request と繋がり、今回の cmd->data データメンバは空になくなったのため、起動するのはデータ伝送である。

6. 実験

デフォルトプラットフォームの情報が変更され

```
.get_ro         = s3cmci_get_ro、
.get_cd         = s3cmci_card_present、

static struct s3c24xx_mci_pdata s3cmci_def_pdata = {
/* This is currently here to avoid a number of if (host->pdata)
* checks. Any zero fields to ensure reaonable defaults are picked. */
.detect_invert=0、
.wprotect_invert=1、
.gpio_detect=1、
.gpio_wprotect = 1 、
};
```

問題：

host_dodma は 1 を設定する場合、/sdcard で内容がない、/proc/devices にも対応デバイスがない

プリントアウト情報から見て：

```
7>mmc0: clock 0Hz busmode 1 powermode 1 cs 0 Vdd 21 width 0 timing 0
<6>s3c2440-sdi s3c2440-sdi: running at 0kHz (requested: 0kHz).
<7>mmc0: clock 197753Hz busmode 1 powermode 2 cs 0 Vdd 21 width 0 timing 0
<6>s3c2440-sdi s3c2440-sdi: running at 198kHz (requested: 197kHz).
<7>mmc0: clock 197753Hz busmode 1 powermode 2 cs 1 Vdd 21 width 0 timing 0
<6>s3c2440-sdi s3c2440-sdi: running at 198kHz (requested: 197kHz).
<7>mmc0: starting CMD0 arg 00000000 flags 000000c0
<7>s3c2440-sdi s3c2440-sdi: CMD[OK] #1 op:0 arg:0x00000000 flags:0x08c0 retries:0
```

```
R0:0x00000000
 <7>mmc0: req done (CMD0): 0: 00000000 00000000 00000000 00000000
 コマンドは成功に送信された、こうなる原因は？

 ここまで、データとコマンドフローに従い、MMC サブシステムの基本構造を分析した。
```

3.18 UDA1341 オーディオドライバを移植

3.18.1 初期化ファイルに UDA1341 デバイス構造を追加

Linux-2.6.32.2 は既に UDA1341 オーディオドライバを十分にサポートし、

```
; ファイルの最初にヘッダファイルを加えます
#include <sound/s3c24xx_uda134x.h>

;LCD プラットフォームデバイス後に UDA1341 デバイス構造を加えます
static struct s3c24xx_uda134x_platform_data s3c24xx_uda134x_data = {
        .l3_clk = S3C2410_GPB(4),
        .l3_data = S3C2410_GPB(3),
        .l3_mode = S3C2410_GPB(2),
        .model = UDA134X_UDA1341,
};

static struct platform_device s3c24xx_uda134x = {
        .name = "s3c24xx_uda134x",
        .dev = {
                .platform_data = &s3c24xx_uda134x_data,
        }
};

; カーネルに UDA1341 デバイスプラットフォームを登録します
static struct platform_device *mini2440_devices[] __initdata = {
        &s3c_device_usb,
        &s3c_device_rtc,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c0,
        &s3c_device_iis,
        &mini2440_device_eth,
        &s3c24xx_uda134x,
        &s3c_device_nand,
};
```

arch/arm/mach-s3c2440/mach-mini2440.c ファイルに　UDA1341 プラットフォームデバイスの制御ポートを登録するだけ、mach-mini2440.c をオープンし、次の内容を追加する

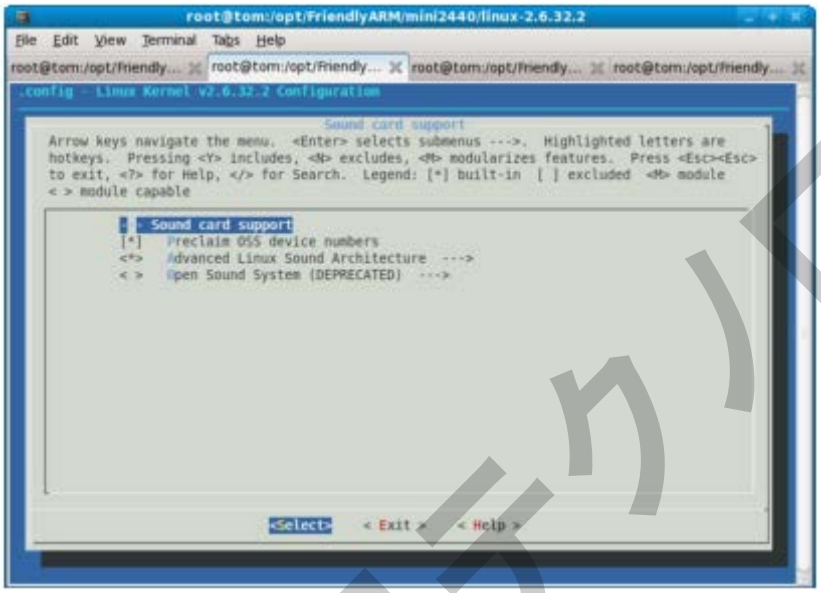
こうして、UDA1341 オーディオドライバを追加できた、その後、カーネルに該当ドライバを設定する

3.18.2　カーネルに UDA1341 ドライバデバイスを設定

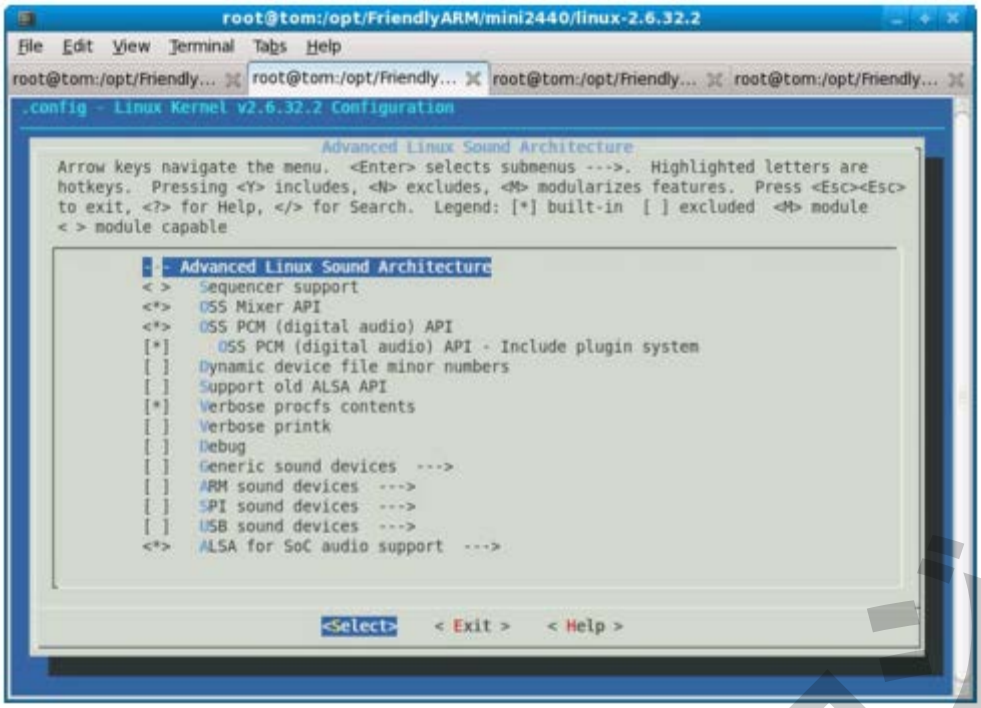
カーネルソースコードディレクトリに make menuconfig を入力し、カーネル設定、次のサブメニューを順に選択し、オーディオドライバ設定メニューを探す

```
Device Drivers  --->
  <*> Sound card support  --->
```

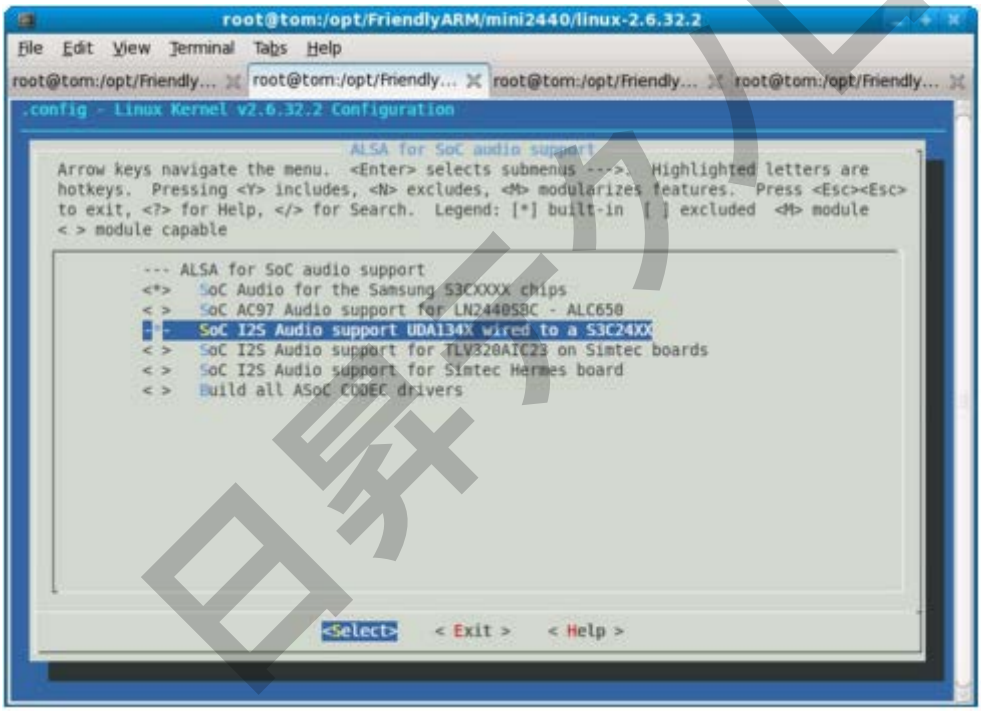
スペースキーを押して、〝[*]　Preclaim OSS device numbers　〟を選択し、また〝<*>　Advanced Linux Sound Architecture　--->〟を選択し、Enter ボタンを押してサブメニューに入る



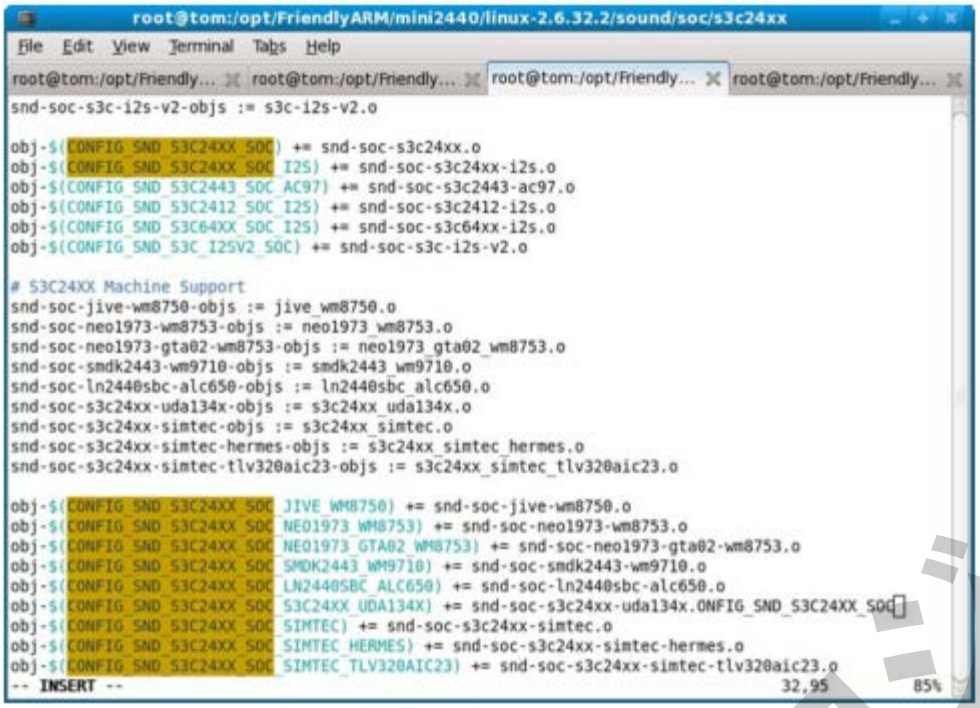
オーディオドライバシステム構造がで、OSS のインターフェイスの設定オプションを選択する。OSS のインターフェイスは ALSA インターフェイスに基づいて作成され、新カーネルには ALSA 設計を変更する、ここで、以前のソフトウェアと相性性がある、下図を参照する



〉〈*〉 ALSA for SoC audio support ---〉サブメニューを選択し、Enter で入る



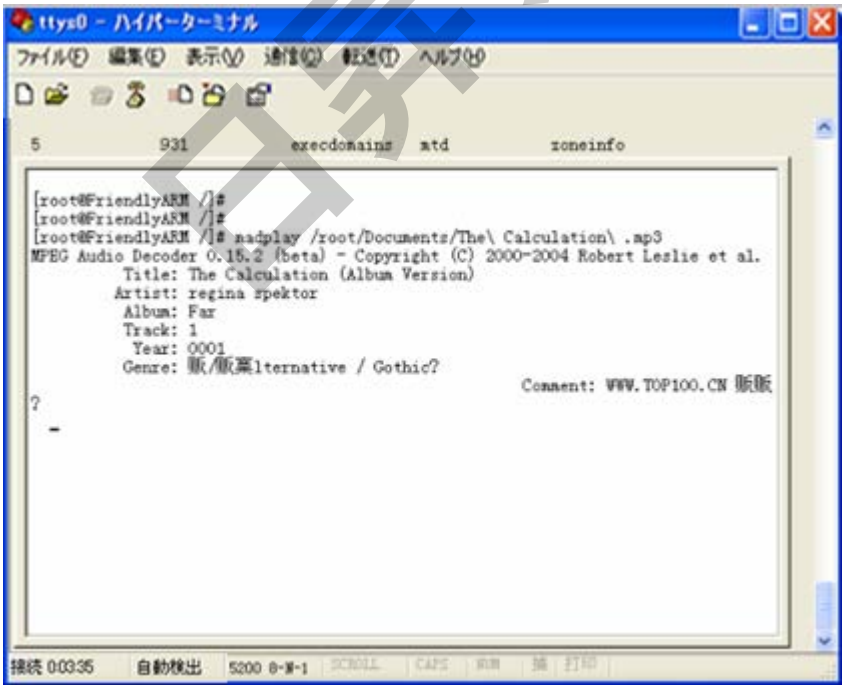
ここで、S3C24xx シリーズチップ(S3C2410/2440/2443 などを含まれる)のため作成される設定オプションを見て linux-2.6.32.2/sound/soc/s3c24xx/Makefile ファイルをオープンする、下図を参照する



開発ボードはUDA1341オーディオチップを使用し、`-*- SoC I2S Audio support UDA134X wired to a S3C24XX`を選択する

3.18.3 MP3再生テスト

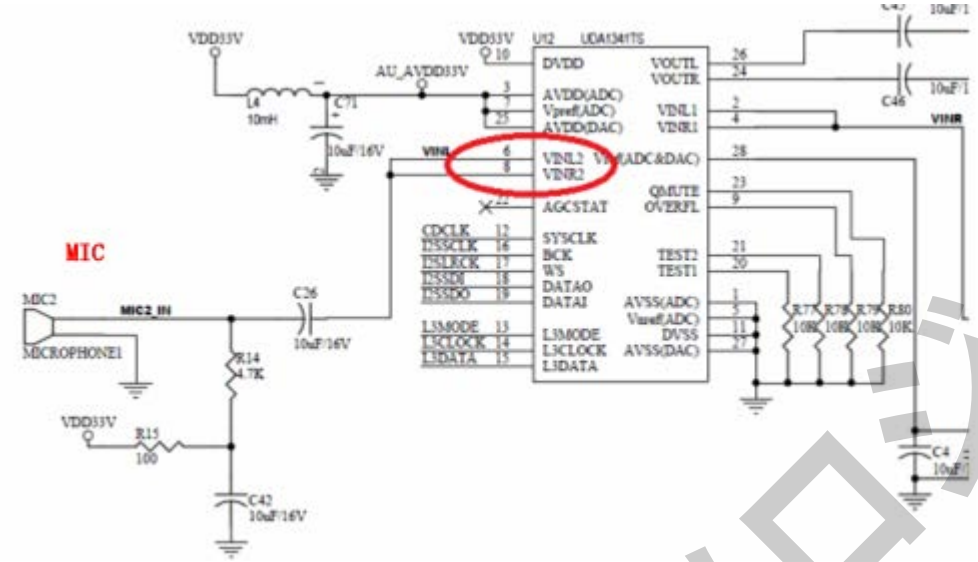
カーネルソースコードディレクトリでmake zImageを実行し、生成したカーネルイメージファイルは開発ボードにプログラミングし、デフォルト・ファイルシステムroot_qtopiaを使用し、システム起動後、madplayソフトウェアを使用し、mp3を再生/テストする、スピーカーまたはヘッドフォンを開発ボードの緑オーディオ出力コンセントに接続。テストを行う、下図を参照する



3.18.4　ドライバでのレコーディングコード修正

ここまで mp3 の再生は正常で行われるが、録音プログラムを使用し、録音の時音声は聞こえなかった、それは開発ボードの録音回路は SMDK2440 のターゲットボードの回路と異なる

Mini2440 開発ボードの録音回路は下図のようになる



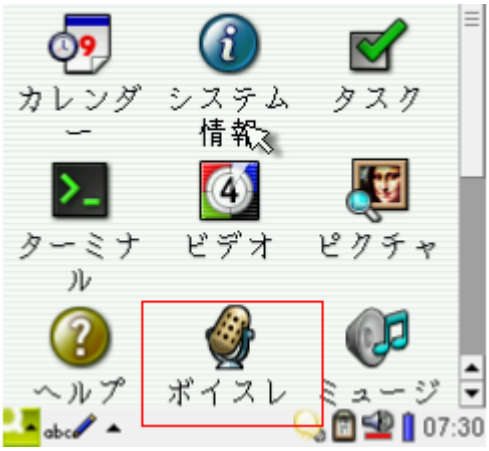
mini2440 開発ボードの録音チャンネルは VIN2 で、SMDK2440 のは VIN1、linux-2.6.32.2/sound/soc/codecs/uda134x.c をオープンし、大体 201 行に次の赤いコードを追加する

```
                uda134x->slave_substream = substream;
        } else
                uda134x->master_substream = substream;
        uda134x_write(codec, 2, 2|(5U<<2)); //録音チャンネルは VIN2 に変更します
            return 0;
}
static void uda134x_shutdown(struct snd_pcm_substream *substream,
            struct snd_soc_dai *dai)
{
```

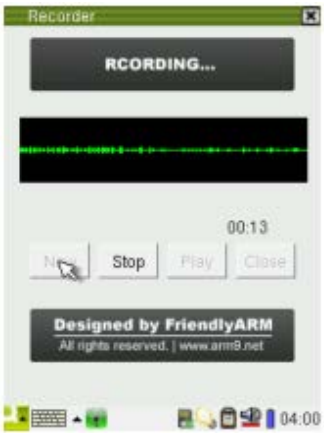
録音ドライバは修正完了、カーネルソースコードディレクトリ下に make zImage を実行し、カーネルをコンバイルし、開発ボードにプログラミング

3.18.5　録音テスト

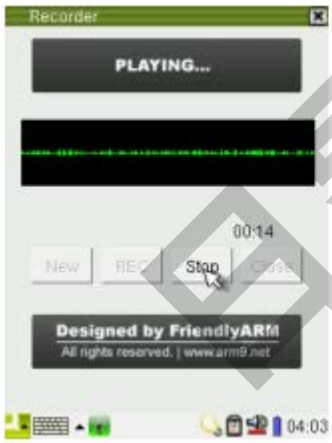
Qtopia の"ボイスレコーダー"のテストプログラムをオープンし、下図を参照する



｀REC｀ボタンをクリックし、録音する、マイクで話し、録音の波形を確認でき、｀STOP｀ボタンで終了、下図を参照する



｀PLAY｀で再生、録音オーディオファイルは｀WAV｀フォーマットでドキュメントに保存される



説明：Qtopia 2.2.0 システムは録音プログラムを持って"ボイスメモ"で録音するが、ハードウェア上ボードのマイクを正常に使用、録音できない。コードのフォーマットを維持するため、修正しなかった。

3.19　シリアルポートドライバを修正

3.19.1　UART2 を一般シリアルポートドライバに変更

S3C2440 チップは 3 つのシリアルがあり、UART0、1、2、ダウンロードした Linux-2.6.32.2 はパーフェク

トな UART0、1 ドライバを備えて、UART2 では赤外線通信(Irda)として使用されるため、普通のシリアルとして使用されるように、UART2 ドライバを修正する必要がある、

先ず、S3C2440 シリアルの部分レジスタの説明を見る、下図のとおり

ULCONn	Bit	Description	Initial State
Reserved	[7]	–	0
Infrared Mode	[6]	Determine whether or not to use the Infrared mode. 0 = Normal mode operation　0　ノーマルモード 1 = Infrared Tx/Rx mode　1　赤外線モード	0
Parity Mode	[5:3]	Specify the type of parity generation and checking during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/checked as 1 111 = Parity forced/checked as 0	000
Number of Stop Bit	[2]	Specify how many stop bits are to be used for end-of-frame signal. 0 = One stop bit per frame 1 = Two stop bit per frame	0
Word Length	[1:0]	Indicate the number of data bits to be transmitted or received per frame. 00 = 5-bits　01 = 6-bits 10 = 7-bits　11 = 8-bits	00

カーネルで UART2 に関する設定を修正し、mach-mini2440.c ファイルをオープンし、次の赤いコードに従い修正する

```
static struct s3c2410_uartcfg mini2440_uartcfgs[] __initdata = {
        [0] = {
                .hwport         = 0,
                .flags          = 0,
                .ucon           = 0x3c5,
                .ulcon          = 0x03,
                .ufcon          = 0x51,
        },
        [1]= {
                .hwport         = 1,
                .flags          = 0,
                .ucon           = 0x3c5,
                .ulcon          = 0x03,
                .ufcon          = 0x51,
},
        /*  UART2 を一般シリアルポートに変更します */
        [2] = {
                .hwport         = 2,
                .flags          = 0,
                .ucon           = 0x3c5,
                .ulcon          = 0x03,
                .ufcon          = 0x51,
        }
};
```

初期化、linux-2.6.32.2/drivers/serial/samsung.c をオーペンし、次の赤いコードを追加する

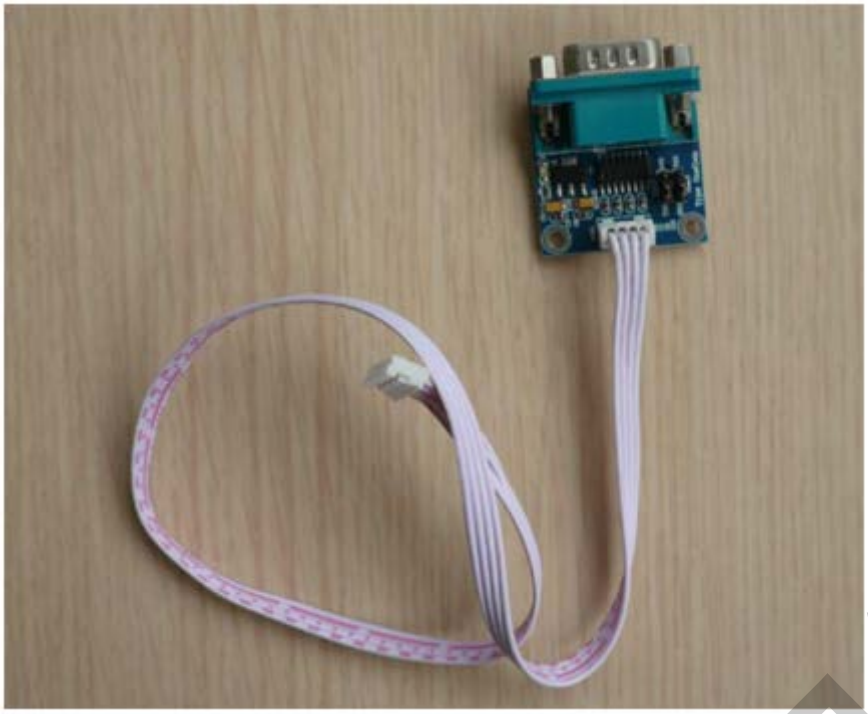
```
//ヘッダファイルを加えます
#include <linux/gpio.h>
#include <mach/regs-gpio.h>

        ourport->tx_claimed = 1;

        dbg("s3c24xx_serial_startup ok¥n");

        /* the port reset code should have done the correct
            * register setup for the port controls */
        //シリアルポート 2 の対応ポート初期化
        if (port->line == 2) {
                s3c2410_gpio_cfgpin(S3C2410_GPH(6), S3C2410_GPH6_TXD2);
                s3c2410_gpio_pullup(S3C2410_GPH(6), 1);
                s3c2410_gpio_cfgpin(S3C2410_GPH(7), S3C2410_GPH7_RXD2);
                s3c2410_gpio_pullup(S3C2410_GPH(7), 1);
        }

        return ret;
```

UART2 修正完了

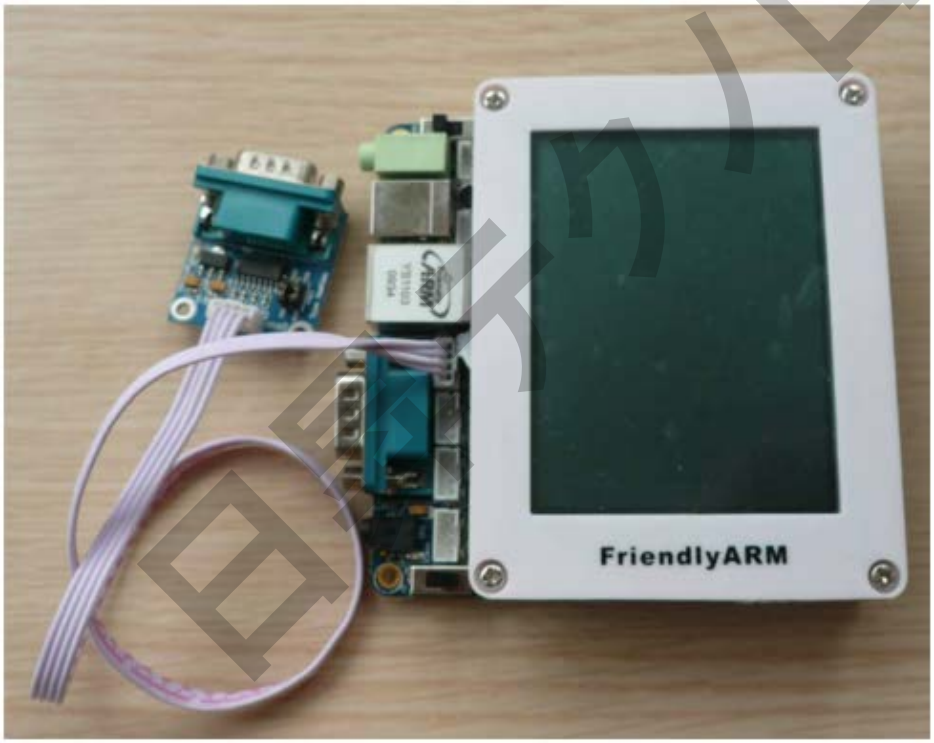
3.19.2　シリアルポートテスト

カーネルソースコードディレクトリに戻り、make zImage を実行し、再度に生成するカーネルは開発ボードにプログラミングし、root_qtopia ファイルシステムを使用し、グラフィカルインターフェースシリアルアシスタントテストプログラムがあるため、テストに便利になる

mini2440 開発ボードは UART2 を RS232 ポートではなく、CON3 ピンを通じて引き出され、RS232 変換ケーブルが必要とする。下図を参照する



mini2440 開発ボードの CON3 インタフェースに接続し、下図を参照する



引き出したシリアルはケーブルを通じて、コンピュータのシリアルに接続し、開発ボードの電源に接続し、S2 を nand 起動に切り替え、通電する。Qtopia システムに入り、シリアルアシスタントをクリック、対応のプログラムインターフェースをオープンする、下図を参照する

プログラムウィンドウのデフォルト設定は｀ttySAC1 115200 8N1 [C]｀で、

- シリアルデバイス：/dev/ttySAC1、シリアル UART1 に対応する

-ボーレート：115200

-データビット：8

- フロー制御：なし

-ストップビット：1

-［C］：文字モードを表示し、[H]は 6 進数モードを表示する

上図には２つのエディットボックスがあり、上記のエディットボックスは受信したデータを表示するに使用され、実際にはコンパイル不可で、下記のエディットボックスは USB ボタンボードや Qtopia のソフトキーボードを通じて取得入力する

UART2 をテストするには、｀Setting｀をクリックし、設定ウィンドウで/dev/ttySAC2 を選択し、メインインターフェイスに戻る、開発ボードシリアル/dev/ttySAC2 をオープンし、Connect をクリック、エディット下のウィンドウに文字を入力し、Send ボタンで接続するシリアルデバイスへデータを送信できる、Windows ハイパーターミナルを通じて、受信したデータの表示である（注：ターミナルに対応するシリアルは 115200 8N1 を設定必要）

3.20 I2C-EEPROM ドライバの移植

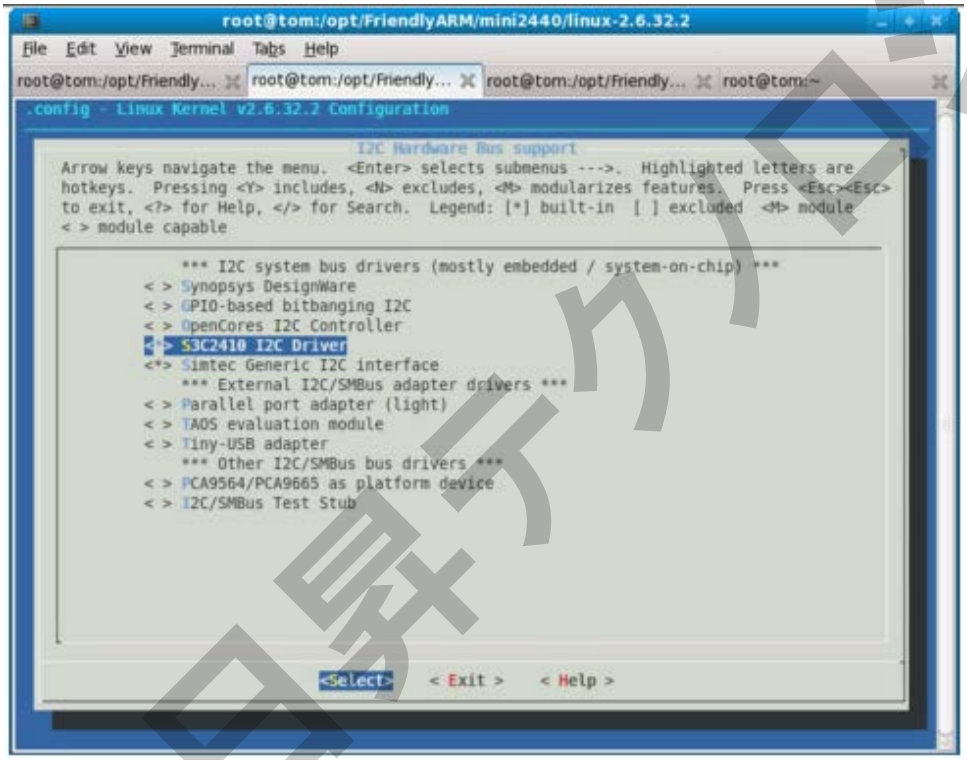
3.20.1 カーネルに I2C ドライバを設定

Linux-2.6.32.2 は S2C2440 の I2C インタフェースにパーフェクトなドライバをサポートしている、カーネルに設定するだけで使用できる

注： Linux-2.6.32.2 カーネル内のデフォルト mini2440_defconfig は既に I2C ドライバを設定済み。カーネルソースコードディレクトリに make menuconfig を実行し、カーネルの設定メインメニューに入り、順に選択し、次のサブメニューに入る

```
Device Drivers  --->
  <*> I2C support  --->
    I2C Hardware Bus support  --->
```

〈*〉S3C2410 I2C Driver を選択、ここの S3C2410 は S3C2440 にも適用し、I2C ポートはレジスタ定義と同じ



設定対応のドライバソースコードは：linux-2.6.32.2/drivers/i2c/busses/i2c-s3c2410.c にある。

3.20.2 I2C-EEPROM をテスト

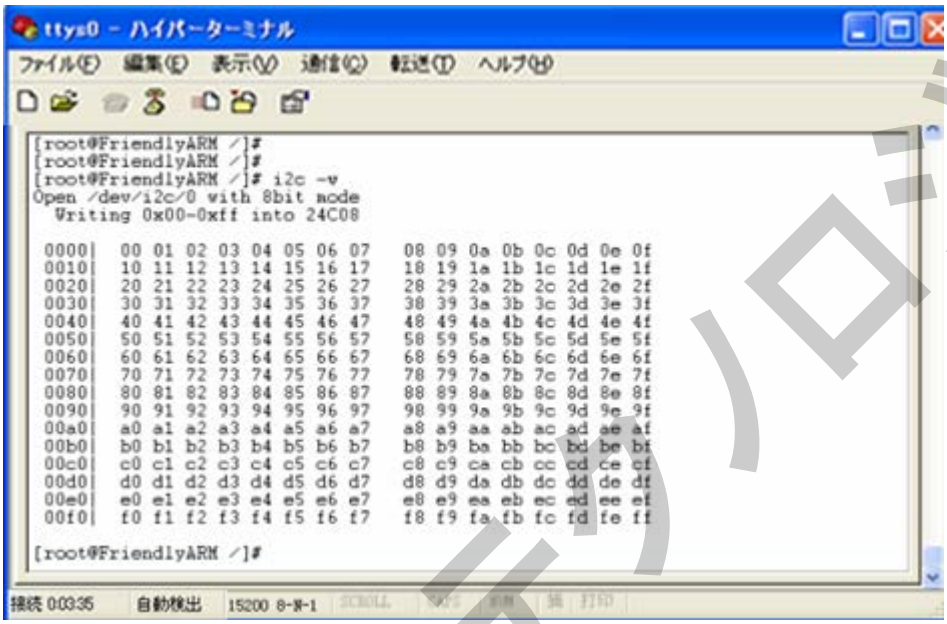
テストプログラム名前：i2c		備考
テストプログラムソースコードファイル名前	Eeprom.c 24cXX.c	
テストプログラムソースコード位置	linux¥examples.tgz を解凍	
クロスコンパイラ	Arm-linux-gcc-4.3.2 with EABI	
開発ボードの対応のデバイス名前	/dev/i2c/0	

対応のカーネルドライバソースコード	Linux-src/drivers/i2c/busses/i2c-s3c2440.c	
その他		

Mini2440 は I2C バスに基づいて EEPROM チップをマウントする、これは AT24C08 で、このチップを書きこみ、読み取ることを通じて、I2C バスドライバをテストできる

カーネルルート･ディレクトリ下に make zImage を実行し、生成されたカーネルは開発ボードにプログラミングし、root_qtopoia を使用する。この中には I2C-EEPROM テストプログラムを含む、コマンドラインとグラフィカル･インターフェースがあり、コマンドラインテストプログラムの名前は｀i2c｀となる。

コマンドライン：i2c -w を入力すると、ボードの 24C08 デバイスデータ（0x00-0xff を書き込まれる



コマンドライン：i2c -r を入力すると、ボードの 24C08 デバイスからの出力を読み出す

3.21　ウォッチドッグドライバ移植

3.21.1 カーネルでウォッチドッグドライバ・コンフィグ

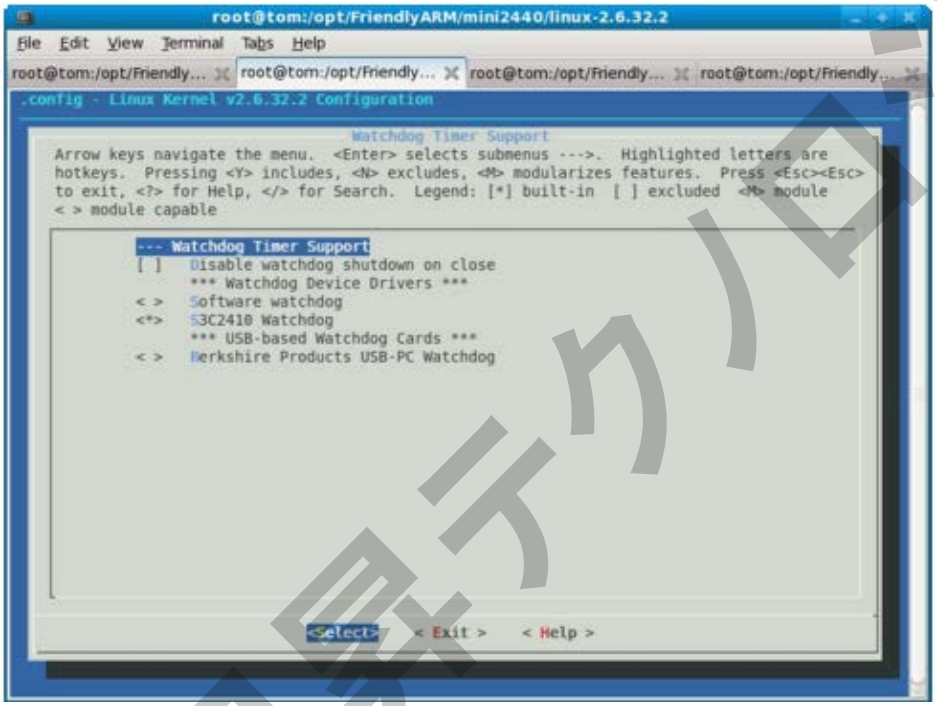
Linux-2.6.32.2 カーネルはパーフェクトな S3C2440 ウォッチドッグドライバがあり、設定するだけで使用できる。

注： Linux-2.6.32.2 カーネルのデフォルトの mini2440_defconfig は既にウォッチドッグドライバを設定した、ここでそれをオープンし、確認できる

カーネルソースコードディレクトリに make menuconfig を実行し、カーネルの設定メインメニューに入り、順に選択し、次のサブメニューに入る

```
Device Drivers  --->
  [*] Watchdog Timer Support  --->
```

ウォッチドッグ設定メニューをオープン、S2C2410/2440 のウォッチドッグ設定オプションを選択する



対応のドライバソースコードは：linux-2.6.32.2/drivers/watchdog/s3c2410_wdt.c にある

3.21.2　ウォッチドッグの ON/OFF について

ウォッチドッグドライバプログラムで赤い部分

```
#define PFX "s3c2410-wdt: "
#define CONFIG_S3C2410_WATCHDOG_ATBOOT (0)
//ウォッチドッグデフォルトの時間は 15 秒で、時間を超えると、システムが自動再起動
#define CONFIG_S3C2410_WATCHDOG_DEFAULT_TIME (15)

static ssize_t s3c2410wdt_write(struct file *file, const char __user *data,
                                size_t len, loff_t *ppos)
{
        /*
        * Refresh the timer.
        */
        if (len) {
                if (!nowayout) {
                        size_t i;

                        /* In case it was set long ago */
                        expect_close = 0;

                        for (i = 0; i != len; i++) {
                                char c;

                                if (get_user(c, data + i))
                                        return -EFAULT;
                                if (c == 'V')
                                        expect_close = 42;
                        }
                }
                s3c2410wdt_keepalive();
        }
        return len;
}
```

ウォッチドッグデバイス(/dev/watchdog をオープンし、ウォッチドッグデバイスに任意なデータを繰り返し書き込み、カウントをクリアし、即ちウォッチドッグにフィードできる。`V` を書き込む時、ウォッチドッグをオフする

3.21.3　ウォッチドッグをテスト

ウォッチドッグの動作は簡単でコードを書けずに直接テストできる。

echo コマンドを使用し、/dev/watchdog へ任意のデータを書き込み、ウォッチドッグを起動する、例えば echo 0 > /dev/watchdog、下図を参照する

15 秒待ち、システムは自動再起動し、この現象によりウォッチドッグが起動した事を明らかにした。

15 秒うちに echo コマンドをループ使用し、ウォッチドッグへデータを書き込む、システムを再起動しない。

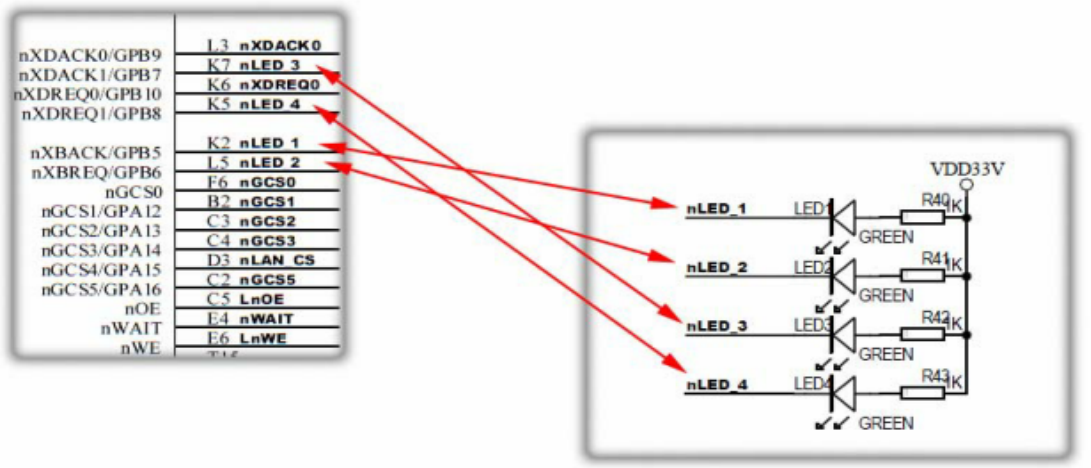
ウォッチドッグを停止する場合、〝V〟を入力するだけ、echo コマンドを使用し、/dev/watchdog へデータを書き込むと同時に、〝Enter〟も書き込まれる、だからこのように操作でき：echo -n V >/dev/watchdog

〝-n〟は〝Enter〟を削除と意味し、テストのため、まず echo 0 > /dev/watchdog を入力し、次に echo -n V > /dev/watchdog を入力し、システムは正常に動いている場合、ウォッチドッグは OFF と証明する

3.22　簡単 LED ドライバ

3.22.1　LED ドライバ原理と書き込み

ドライバを書くには関連のハードウェアを理解する必要があり、例えば、使用されるレジスタ、物理アドレス、割り込みなど、mini2440 原理図には LED のハードウェア接続は次の通り



S3C2440 のデータシートを調べて、対応のハードウェアリソースは下記の通り

LED 番号	対応の GPIO	対応の CPU ピン
LED1	GPIO5	K2
LED2	GPIO6	L5
LED3	GPIO7	K7
LED4	GPIO8	K5

GPIO は汎用の入出力ポートの略称であり、S3C2440 チップに、ポートは再利用し、GPIO はその 1 つの機能で、例えばここの GPB5 ポートは普通の GPIO として使用され、専用機能の nXBACK に使用される、ピンの機能を変更できるように、該当のポートレジスタを設定する必要がある。

四つ LED は GPBCON レジスタの四組 2bit ビットを使用し、対応ピンの用途を設定する。四組 2bit ビットの機能は同じで、00 は入力、01 は出力のことを現れ、10 は特殊機能、11 は保持 、詳細は下記の通り：

PBCON	Bit	Description
GPB10	[21:20]	00 = Input 01 = Output 10 = nXDREQ0 11 = reserved
GPB9	[19:18]	00 = Input 01 = Output 10 = nXDACK0 11 = reserved
GPB8	[17:16]	00 = Input 01 = Output 10 = nXDREQ1 11 = Reserved
GPB7	[15:14]	00 = Input 01 = Output 10 = nXDACK1 11 = Reserved
GPB6	[13:12]	00 = Input 01 = Output 10 = nXBREQ 11 = reserved
GPB5	[11:10]	00 = Input 01 = Output 10 = nXBACK 11 = reserved
GPB4	[9:8]	00 = Input 01 = Output 10 = TCLK [0] 11 = reserved
GPB3	[7:6]	00 = Input 01 = Output 10 = TOUT3 11 = reserved
GPB2	[5:4]	00 = Input 01 = Output 10 = TOUT2 11 = reserved]
GPB1	[3:2]	00 = Input 01 = Output 10 = TOUT1 11 = reserved
GPB0	[1:0]	00 = Input 01 = Output 10 = TOUT0 11 = reserved

mini2440 開発ボードに、

LED1 は GPB5 を対応し、 GPB5 は［11:10］ビットを使用する

LED2 は GPB6 を対応し、 GPB6 は［12:13］ビットを使用する
LED3 は GPB7 を対応し、 GPB7 は［14:15］ビットを使用する
LED4 は GPB8 を対応し、 GPB8 は［16:17］ビットを使用する
ドライバプログラムに LED は出力状態を設定される必要があり、即ち、GPBX は 01 を設定される

GPBDAT レジスタは四つ LED の数値状態を対応するに使用され、GPBDAT5 は GPB5 を対応し、GPBDAT6 は GPB6 を対応する…回路図によると、GPIO 出力はローレベルの時、有効、即ち、レジスタは 0 の位置になる時、GPB5、6、7、8 はローレベルを出力し、対応の LED は点灯する

ソフトウェアに使用される IO ポートを操作するには一般に既存の関数やマクロを呼び出す、例えば s3c2410_gpio_cfgpin、どうして S3C2410 であるか、サムスンの生産する S3C2440 チップが使用するレジスタ名前とリソースの割り当ては S3C2410 と殆ど同じですから、現在の各バージョンの Linux システムに大体同じ関数定義とマクロ定義を使用する
linux-2.6.32.2/arch/arm/plat-s3c24x/gpio.c ファイルに該当関数の定義と実現を見つけ関連定義できる。当関数の名前は一般変更しないので、他のドライバソースコードに該当関数を含めるヘッダーファイルで見つける。gpio.c ファイルに s3c2410_gpio_cfgpin 関数の実現は下記の通り

```
void s3c2410_gpio_cfgpin(unsigned int pin, unsigned int function)
{
        void __iomem *base = S3C24XX_GPIO_BASE(pin);
        unsigned long mask;
        unsigned long con;
        unsigned long flags;

        if (pin < S3C2410_GPIO_BANKB) {
                mask = 1 << S3C2410_GPIO_OFFSET(pin);
        } else {
                mask = 3 << S3C2410_GPIO_OFFSET(pin)*2;
        }

        switch (function) {
        case S3C2410_GPIO_LEAVE:
                mask = 0;
                function = 0;
                break;

        case S3C2410_GPIO_INPUT:
        case S3C2410_GPIO_OUTPUT:
        case S3C2410_GPIO_SFN2:
        case S3C2410_GPIO_SFN3:
                if (pin < S3C2410_GPIO_BANKB) {
                        function -= 1;
                        function &= 1;
                        function <<= S3C2410_GPIO_OFFSET(pin);
                } else {
                        function &= 3;
                        function <<= S3C2410_GPIO_OFFSET(pin)*2;
                }
        }

        /* modify the specified register wwith IRQs off */

        local_irq_save(flags);

        con = __raw_readl(base + 0x00);
```

```
        con &= ~mask;
        con |= function;

        __raw_writel(con, base + 0x00);

        local_irq_restore(flags);
}
```

次のドライバプログラムリストに s3c2410_gpio_cfgpin の呼び出し情況を確認できる。他のデバイスドライバに関連する基本関数の呼び出し、例えば登録デバイス misc_register、ドライバ関数を書く構造 file_operations と Hello、Module（ユーザーマニュアルの例を参照する）のような module_init と module_exit 関数など。

drivers/char ディレクトリ下にドライバプログラムファイルを作成し、内容は下記の通り

```
#include <linux/miscdevice.h>
#include <linux/delay.h>
#include <asm/irq.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/delay.h>
#include <linux/moduleparam.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/ioctl.h>
#include <linux/cdev.h>
#include <linux/string.h>
#include <linux/list.h>
#include <linux/pci.h>
#include <linux/gpio.h>
#include <asm/uaccess.h>
#include <asm/atomic.h>
#include <asm/unistd.h>
```

```
#define DEVICE_NAME "leds" //デバイス名(/dev/leds)

//LED 対応の  GPIO ポートリスト
static unsigned long led_table [] = {
    S3C2410_GPB(5),
    S3C2410_GPB(6),
    S3C2410_GPB(7),
    S3C2410_GPB(8),
};

//LED 対応ポート出力のステータスリスト
static unsigned int led_cfg_table [] = {
    S3C2410_GPIO_OUTPUT,
    S3C2410_GPIO_OUTPUT,
    S3C2410_GPIO_OUTPUT,
    S3C2410_GPIO_OUTPUT,
};

/*ioctl  関数の実現
 * アプリレイヤ/ユーザーレイヤは  ioctl  関数でカーネルへパラメータを伝送し、LED  の出力状態を制御する
 */
static int sbc2440_leds_ioctl(
    struct inode *inode,
    struct file *file,
    unsigned int cmd,
    unsigned long arg)
{
switch(cmd) {
    case 0:
    case 1:
        if (arg > 4) {
            return -EINVAL;
        }
        //アプリレイヤ/ユーザーレイヤからのパラメータ（否定演算子する）により、s3c2410_gpio_setpin 関数で LED の対応ボードレジスタを設定します。
        s3c2410_gpio_setpin(led_table[arg], !cmd);
        return 0;
    default:
        return -EINVAL;
```

```
    printk (DEVICE_NAME"¥tinitialized¥n"); //初期化情報プリントアウト

    return ret;
}

static void __exit dev_exit(void)
{
    misc_deregister(&misc);
}

module_init(dev_init); //モジュール初期化、コマンド insmod/podprobe でロードする場合だけで有効します、他の方法でロードする場合は呼び出ししません
module_exit(dev_exit);// モジュール・アンインストール、デバイスはモジュール方式でロードした後、コマンド rmmod でアンインストールできます
MODULE_LICENSE("GPL"); //著作権情報
MODULE_AUTHOR("FriendlyARM Inc."); //開発者情報
```

説明：その他デバイス（misc device）

他デバイスは組み込みシステムの汎用バイスドライバであり、Linux カーネルの include/linux ディレクトリ下に Miscdevice.h ファイルがあり、自分定義の misc device サブデバイスはここで定義される。

次に、LED デバイスのカーネル設定オプションを追加する、drivers/char/Kconfig をオープンし、以下の赤い部分を追加する：

```
config DEVKMEM
        bool "/dev/kmem virtual device support"
        default y
        help
          Say Y here if you want to support the /dev/kmem device. The
          /dev/kmem device is rarely used, but can be used for certain
          kind of kernel debugging operations.
          When in doubt, say "N".

config LEDS_MINI2440
        tristate "LED Support for Mini2440 GPIO LEDs"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
```

```
        help
          This option enables support for LEDs connected to GPIO lines
          on Mini2440 boards.

config MINI2440_ADC
        bool "ADC driver for FriendlyARM Mini2440 development boards"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
        help
          this is ADC driver for FriendlyARM Mini2440 development boards
          Notes: the touch-screen-driver required this option
```

次に、ドライバの設定定義に基づき、対応するドライブのターゲット･ファイルをカーネルに追加する。linux-2.6.32.2/drivers/char/Makefile ファイルをオープンし、次のように赤い部分を追加する：

```
obj-$(CONFIG_JS_RTC) += js-rtc.o
js-rtc-y = rtc.o

obj-$(CONFIG_LEDS_MINI2440) += mini2440_leds.o
obj-$(CONFIG_MINI2440_ADC) += mini2440_adc.o

# Files generated that shall be removed upon make clean
clean-files := consolemap_deftbl.c defkeymap.c
```

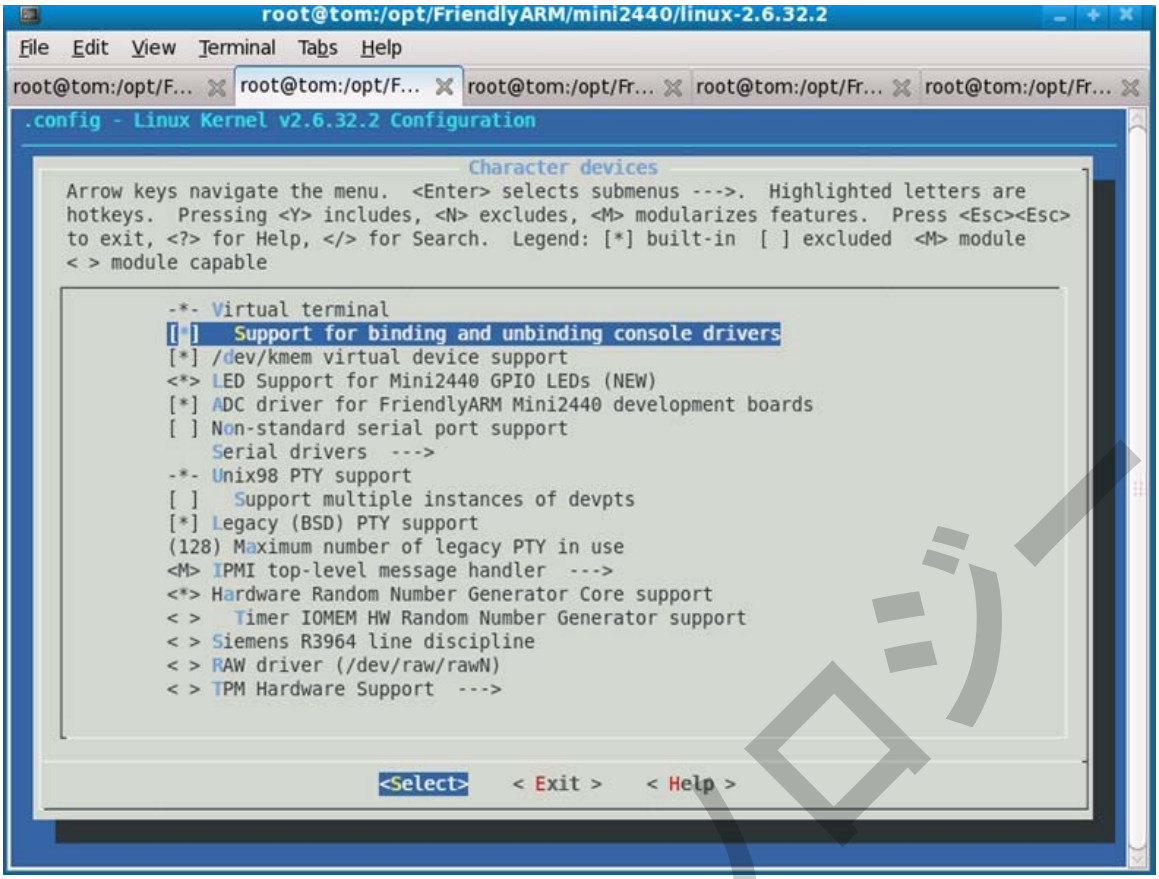
こうして、カーネルに LED ドライバを追加した。

3.22.2 新カーネルコンパイルと LED テスト

上記の手順で、カーネルのソースコード･ディレクトリで下記のコマンドを実行する：make menuconfig カーネル再コンパイル、サブメニュに入る：

```
Device Drivers  --->
  Character devices  --->
```

LED ドライバの設定メニューに入る：



`〈*〉 LED Support for Mini2440 GPIO LEDs (NEW)` オプションを選択、上記のカーネル設定を保存し、終了する。

カーネルのソースコードのディレクトリで次のコードを実行：；make zImage、生成された新カーネルを開発ボードにプログラミングする。

3.22.3 LED テスト

ドライバをテストするため、テストプログラムをコンパイルする必要がある、ドライバ内 ioctl 関数を呼び出して、LED を制御する、leds-test.c ファイルを生成し、コード：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>

int main(int argc, char **argv)
{
        int on;
        int led_no;
        int fd;
        if (argc != 3 || sscanf(argv[1], "%d", &led_no) != 1 || sscanf(argv[2],"%d", &on) != 1 ||
                on < 0 || on > 1 || led_no < 0 || led_no > 3) {
```

```
                fprintf(stderr, "Usage: leds led_no 0|1¥n");
                exit(1);
        }
        fd = open("/dev/leds0", 0);
        if (fd < 0) {
                fd = open("/dev/leds", 0);
        }
        if (fd < 0) {
                perror("open device leds");
                exit(1);
        }
        ioctl(fd, on, led_no);
        close(fd);
        return 0;
}
```

付属 DVD でテストプログラムのソースコードがあり、¥linux¥examples¥leds ディレクトリにある、ファイル名は led.c。

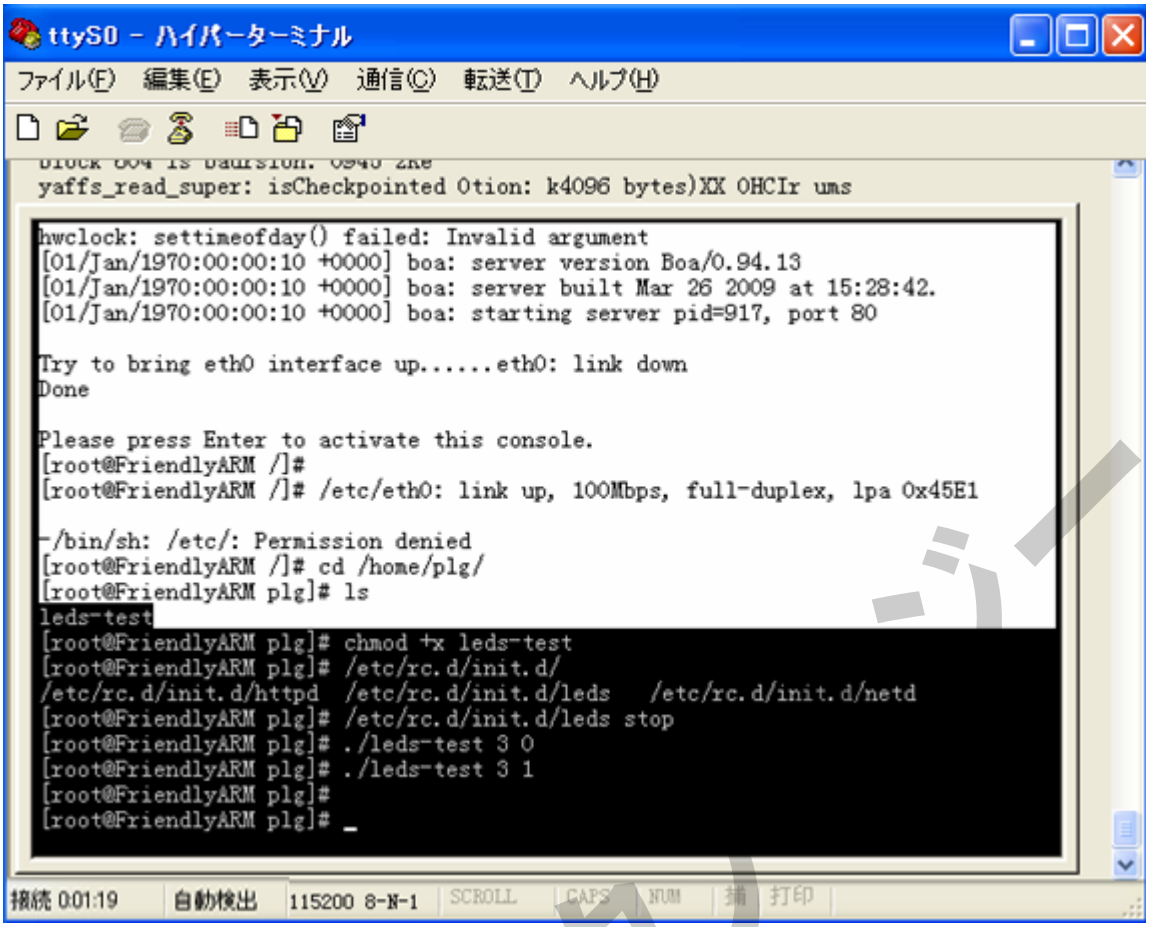
コマンドライン：#arm-linux-gcc -o leds-test leds-test.c

実行可能なオブジェクト･ファイル leds-test を生成され、ネット ftp または USB メモリードライブで開発ボードにプログラミングし、/home/plg ディレクトリ(注：開発ボードのデフォルトファイルシステムには既に LED テストプログラムがあるので、ファイル名を leds-test に変更する)、開発ボードでコマンドライン：#/etc/rc.d/init.d/leds stop を実行する。led-player から led の制御を停止する機能である、led-player についてはユーザーマニュアルを参照する。leds-test で led の制御方法は：

#leds-test 3 0 ； LED3 オフ

#leds-test 3 1 ； LED3 オン

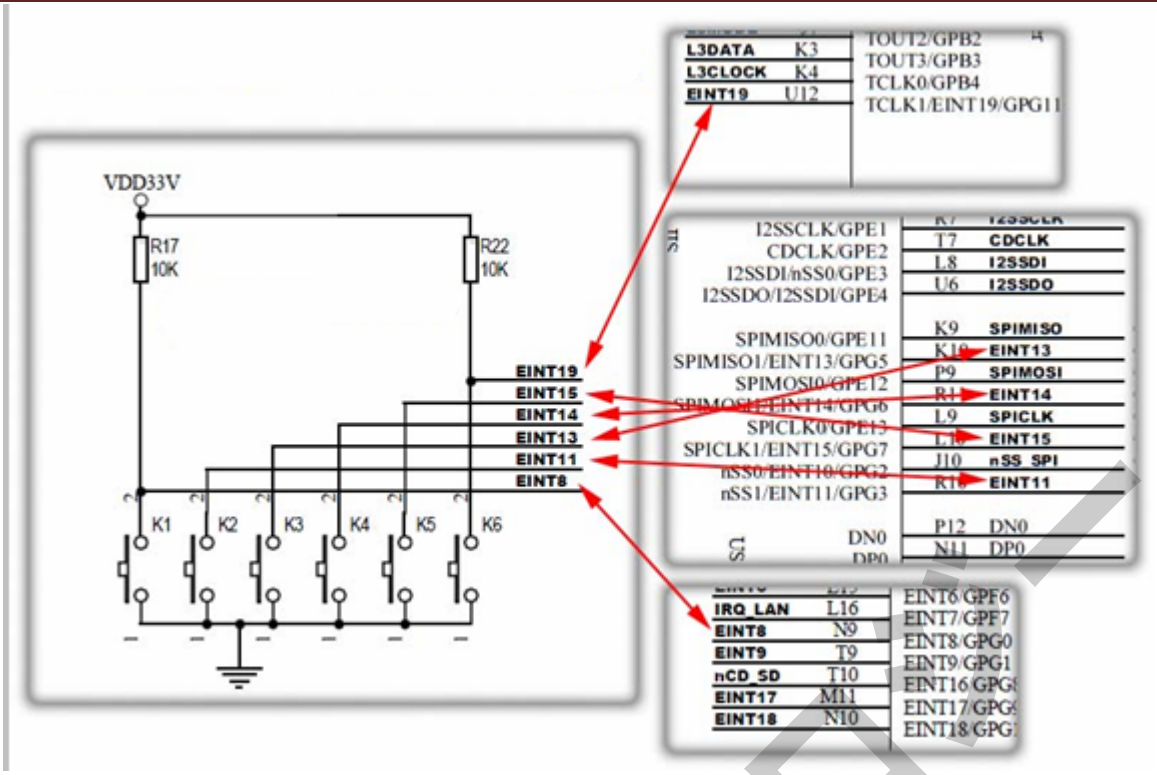
1 つのパラメータは制御する LED のシリアル番号、2 番目のパラメータはオフ（0）またはオン（1）に対応する LED を表示する。下図を参照する:



3.23 割り込みに基づきのボタンドライバ

3.23.1 ハードウェア回路図

Mini2440 には 6 つのユーザーテストボタンがあり、すべて CPU の割り込みピンに接続する：



6 つのユーザーテストボタンと CPU の割り込みピンの対応関係：

ボタン	対応ポートレジスタ	対応の割り込み	対応の多重化機能
K1	GPG0	EINT8	GPIO と割り込み機能のみ
K2	GPG3	EINT11	nSS1
K3	GPG5	EINT13	SPIMISO
K4	GPG6	EINT14	SPIMOSI
K5	GPG7	EINT15	SPICLK
K6	GPG11	EINT19	TCLK

先ず、割り込み機能があり、直接割り込みを実験できる。中に GPG3、5、6、7 の組み合わせは SPI インターフェースを構成できる。そして便利にするように CON12 を増やし、これらのボタンをパネルと繋がり、または拡張・フル機能キーボードインタフェースとして使用する。これも mini2440 の特性の一つである。

3.23.2 ドライバプログラム分析とコンパイル

/linux-2.6.32.2/drivers/char/ディレクトリで新しいドライバプログラム mini2440_buttons.c を作成する：

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/poll.h>
#include <linux/irq.h>
#include <asm/irq.h>
#include <linux/interrupt.h>
#include <asm/uaccess.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/platform_device.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>
#include <linux/sched.h>
#include <linux/gpio.h>

#define DEVICE_NAME             "buttons" //デバイス名
```

```
/*割り込み用の構造体を定義する*/
struct button_irq_desc {
    int irq; //ボタンの割り込みナンバー
    int pin; /ボタンの GPIO ポート
    int pin_setting; //ボタンのピン設定、実際使用しない、保留
    int number; //ボタン値定義、アプリケーション層/ユーザー状態に伝送
    char *name; //各ボタンの名前
};

/*構造体の定義*/
static struct button_irq_desc button_irqs [] = {
    {IRQ_EINT8 、 S3C2410_GPG(0) 、 S3C2410_GPG0_EINT8 、 0、 "KEY0"}、
    {IRQ_EINT11、 S3C2410_GPG(3) 、 S3C2410_GPG3_EINT11 、 1、 "KEY1"}、
    {IRQ_EINT13、 S3C2410_GPG(5) 、 S3C2410_GPG5_EINT13 、 2、 "KEY2"}、
    {IRQ_EINT14、 S3C2410_GPG(6) 、 S3C2410_GPG6_EINT14 、 3、 "KEY3"}、
    {IRQ_EINT15、 S3C2410_GPG(7) 、 S3C2410_GPG7_EINT15 、 4、 "KEY4"}、
    {IRQ_EINT19、 S3C2410_GPG(11)、 S3C2410_GPG11_EINT19、 5、 "KEY5"}、
};

/*開発ボードのボタンの状態変数、ここは゛0゛、該当の ASCII コードは 30*/

static volatile char key_values [] = {'0'、 '0'、 '0'、 '0'、 '0'、 '0'};
/*ドライブは割り込みに基づいているので、ここで待機中のキューが作成され、割り込み関数と合わせます；
ボタンが押されて、ボタン値を読み込まれたときに、キューにはウェイクアップし、割り込みフラグをセット
する。read 関数を介して判断し、ボタンを読み取りユーザーモードに伝送する。ボタンが押されていない場合、
システムがボタンの状態をポーリングしません、クロック･リソースを節約する*/
static DECLARE_WAIT_QUEUE_HEAD(button_waitq);

/*割り込み識別変数、上記のキューの使用と合わせます、割り込みサービス･ルーチンはそれを 1 に設定する。
read 関数はそれをクリアする*/
static volatile int ev_press = 0;

/*ボタンドライバーの割り込みサービス･ルーチン*/
static irqreturn_t buttons_interrupt(int irq、 void *dev_id)
{
struct button_irq_desc *button_irqs = (struct button_irq_desc *)dev_id;
int down;
// udelay(0);
/*ボタンが押された状態を取得*/
```

```
    down = !s3c2410_gpio_getpin(button_irqs->pin);

    /*状態変更、ボタンが押されていない場合、レジスタの値は 1(プルアップ)、ボタンが押されたとき、レジスタの値は 0*/
    if (down != (key_values[button_irqs->number] & 1)) { // Changed

    /* key1 が押されたとき、 key_value[0]は'1'、該当の ASCII コードは 31*/
    key_values[button_irqs->number] = '0' + down;

        ev_press = 1; /*割り込みフラグを 1 に設定*/
        wake_up_interruptible(&button_waitq); /*待機キュー・ウェイクアップ*/
    }
    return IRQ_RETVAL(IRQ_HANDLED);
}

/*
*プログラムが open("/dev/buttons"、...)を実行する時に呼び出す、ここでの機能は 6 つボタンの割り込みを登録する。
*割り込みタイプは IRQ_TYPE_EDGE_BOTH、デュアルエッジ・トリガで、立ち上がりエッジと立ち下がりエッジ共に割り込みが発生する。これはボタンの状態をよく正確に判断することです。
*/
static int s3c24xx_buttons_open(struct inode *inode, struct file *file)
{
    int i;
    int err = 0;

    for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++) {
    if (button_irqs[i].irq < 0) {
        continue;
    }
        /*割り込み関数登録*/
        err = request_irq(button_irqs[i].irq, buttons_interrupt, IRQ_TYPE_EDGE_BOTH,
                          button_irqs[i].name, (void *)&button_irqs[i]);
        if (err)
            break;
    }
```

```
    if (err) {
        /*エラーが出ると、登録した割り込みをアンインストール、戻る*/
        i--;
        for (; i >= 0; i--) {
        if (button_irqs[i].irq < 0) {
        continue;
        }
        disable_irq(button_irqs[i].irq);
            free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
        }
        return -EBUSY;
    }
    /*登録成功すると、割り込みキューは 1 を標識、 read で読み取る*/
    ev_press = 1;

    /*正常戻り*/
    return 0;
}
/*
*この関数がシステム呼び出し関数 close(fd)と対応、ここでの機能は 6 つボタンの割り込みをアンインストールする。
*/
static int s3c24xx_buttons_close(struct inode *inode, struct file *file)
{
    int i;
    for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++) {
    if (button_irqs[i].irq < 0) {
        continue;
    }
    /*割り込みナンバーアンインストール、割り込みハンドラ関数をキャンセル*/
    free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
    }

    return 0;
}
```

```
/*
*プログラムの read(fd,…)関数と対応、機能はボタン値をユーザスペースに伝送する
*/
static int s3c24xx_buttons_read(struct file *filp, char __user *buff, size_t count, loff_t *offp)
{
unsigned long err;

    if (!ev_press) {
    if (filp->f_flags & O_NONBLOCK)
        /*割り込みフラグは 0 で、デバイスが非ブロッキングオープンされた場合、戻り*/
        return -EAGAIN;
    else
        /*割り込みフラグは 0 で、デバイスがブロッキングオープンされた場合、休止状態に入り、ウェイクアップ待ち */
        wait_event_interruptible(button_waitq, ev_press);
}

    /*割り込みフラグクリア*/
    ev_press = 0;

    /* 1 セットボタン値はユーザー· スペースに伝送される*/
    err = copy_to_user(buff, (const void *)key_values, min(sizeof(key_values), count));
    return err ? -EFAULT : min(sizeof(key_values), count);
}
static unsigned int s3c24xx_buttons_poll( struct file *file, struct poll_table_struct *wait)
{
    unsigned int mask = 0;
    /* poll または select を呼び出したプロセスをキューにハングし、ドライバ・ウェイクアップ待ち*/
    poll_wait(file, &button_waitq, wait);
    if (ev_press)
        mask |= POLLIN | POLLRDNORM;
    return mask;
}
/*デバイス操作セット*/
static struct file_operations dev_fops = {
    .owner   =   THIS_MODULE,
    .open    =   s3c24xx_buttons_open,
    .release   =   s3c24xx_buttons_close,
    .read    =   s3c24xx_buttons_read,
    .poll    =   s3c24xx_buttons_poll,
```

```
static struct miscdevice misc = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DEVICE_NAME,
    .fops = &dev_fops,
};

/*デバイス初期化、主はデバイスの登録*/
static int __init dev_init(void)
{

    int ret;

    /*ボタンデバイスを misc デバイスとして登録し、デバイスナンバーは自動割り当てられた*/
    ret = misc_register(&misc);

    printk (DEVICE_NAME"¥tinitialized¥n");

    return ret;
}

/*デバイス・アンインストール*/
static void __exit dev_exit(void)
{
    misc_deregister(&misc);
}

module_init(dev_init); //モジュール初期化、コマンド insmod/podprobe でロードする場合だけで有効する、他の方法でロードする場合は呼び出さない
module_exit(dev_exit); //モジュール・アンインストール、デバイスはモジュール方式でロードした後、コマンド rmmod でアンインストールできる


MODULE_LICENSE("GPL");//著作権情報
MODULE_AUTHOR("FriendlyARM Inc.");// /開発者情報
```

3.23.3 ボタンドライバをカーネルに追加

ボタンドライバをカーネルに加えます、 linux-2.6.32.2_fa/drivers/char/Kconfig をオープンし、赤字のコードを追加する：

```
config MINI2440_BUTTONS
        tristate "Buttons driver for FriendlyARM Mini2440 development boards"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
        help
            this is buttons driver for FriendlyARM Mini2440 development boards

config MINI2440_BUZZER
        tristate "Buzzer driver for FriendlyARM Mini2440 development boards"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
        help
            this is buzzer driver for FriendlyARM Mini2440 development boards

config MINI2440_ADC
        bool "ADC driver for FriendlyARM Mini2440 development boards"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
        help
        this is ADC driver for FriendlyARM Mini2440 development boards
        Notes: the touch-screen-driver required this option

config BFIN_JTAG_COMM
        tristate "Blackfin JTAG Communication"
```

次に、ドライバ構成定義に基づき、ドライブのターゲットファイルをカーネルに追加する、linux-2.6.32.2/drivers/char/Makefile をオープンし、赤字のコードを追加する：

```
obj-$(CONFIG_JS_RTC)                    += js-rtc.o
js-rtc-y = rtc.o

obj-$(CONFIG_LEDS_MINI2440)             += mini2440_leds.o
obj-$(CONFIG_MINI2440_BUTTONS) += mini2440_buttons.o
obj-$(CONFIG_MINI2440_ADC)    += mini2440_adc.o

# Files generated that shall be removed upon make clean
clean-files := consolemap_deftbl.c defkeymap.c
```

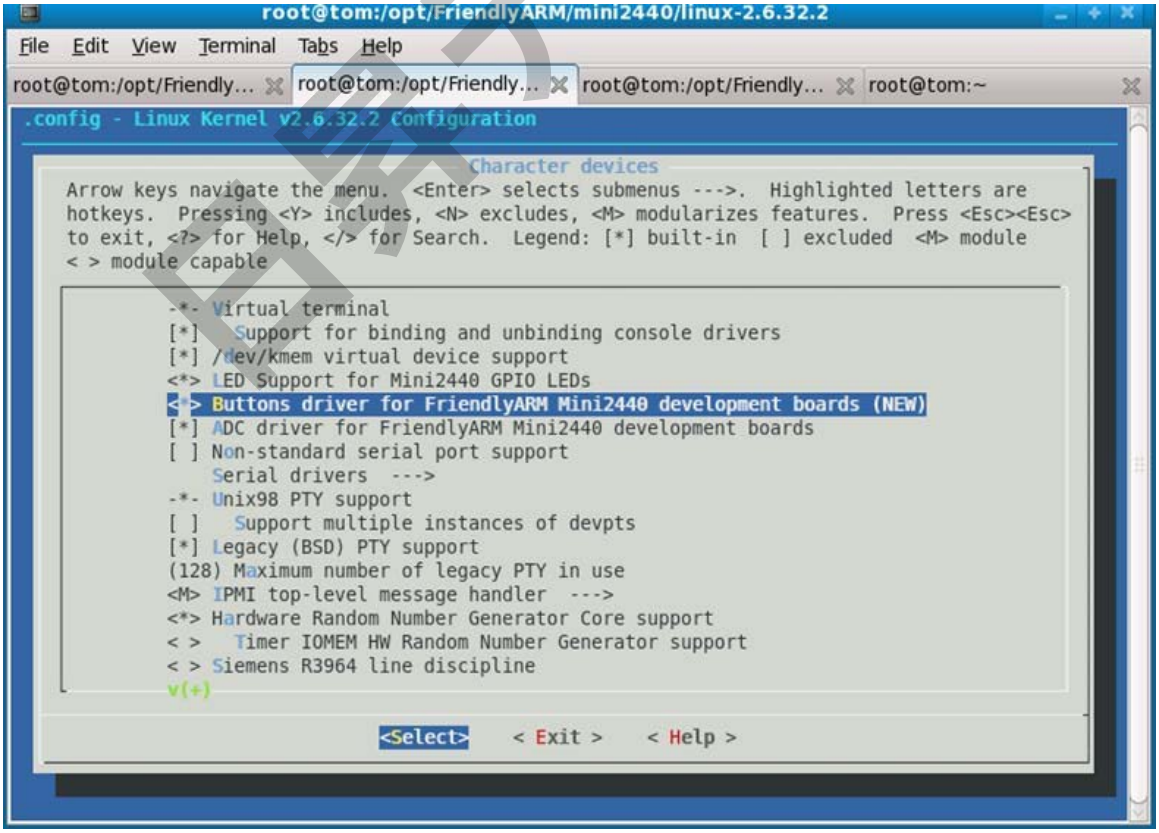
ここまで、カーネルに LED ドライバは追加した

3.23.4　カーネル再構成・コンフィグ

上記の手順で、カーネルのソースコードのディレクトリで実行：make menuconfig カーネルを再構成、サブメニュに入る：

```
Device Drivers --->
 Character devices --->
```

ボタンドライバの設定メニューに入る：



スペースボタンで〈*〉Buttons driver for FriendlyARM Mini2440 development boards(NEW)〉 オプションを選択、上記のカーネル設定を保存し、終了する。

カーネルのソースコードのディレクトリで次のコマンドを実行：make zImage、生成された新カーネルを開発ボードにプログラミングする。

3.23.5 ボタンテスト

前章のボタンをテストするため、テストプログラムを作成する：btn-test.c：

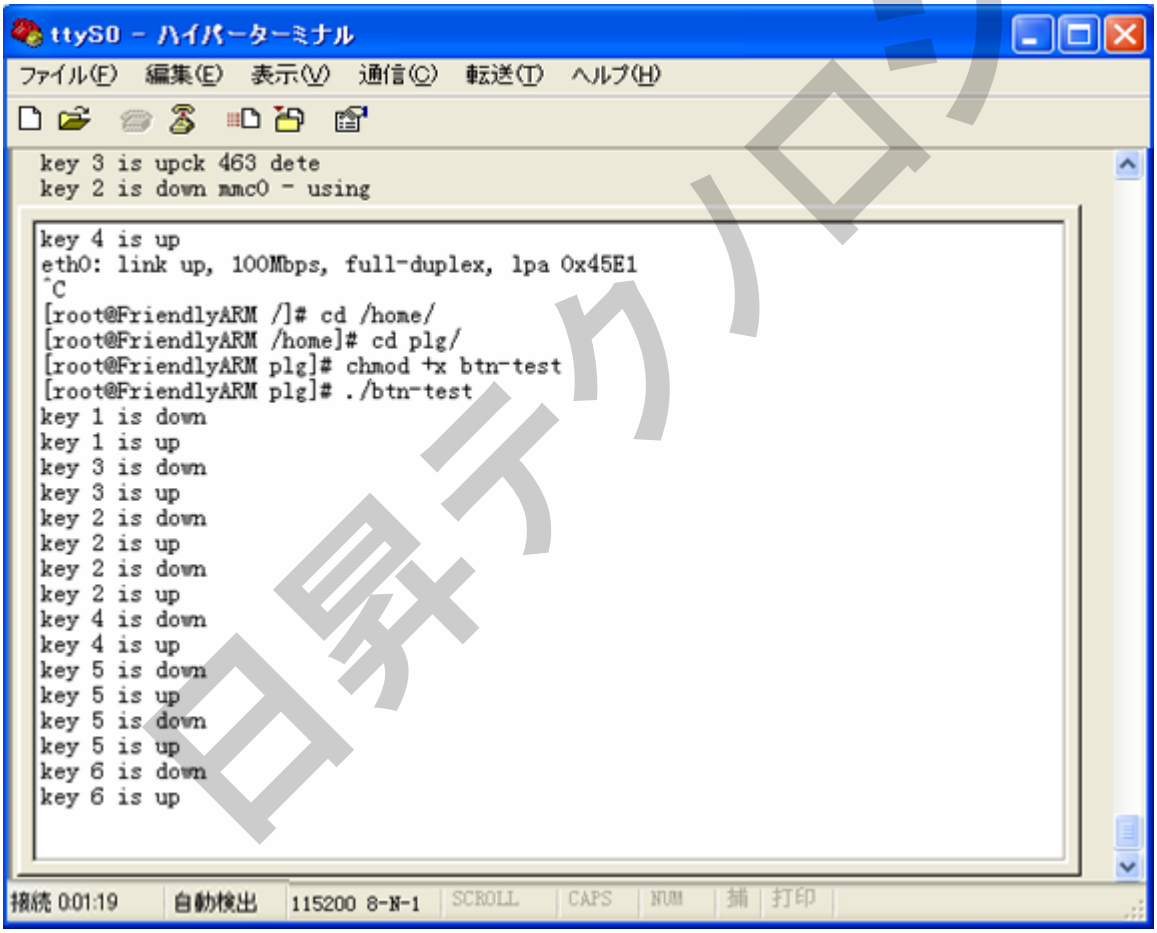
注：付属 DVD にテストプログラムのソースコードがある。位置は：¥linux¥examples¥buttons 、ファイル名：buttons_test.c、ここでは名前を btn-test.c に変更する。

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>

int main(void)
```

```
{
    int buttons_fd;
    char buttons[6] = {'0', '0', '0', '0', '0', '0'}; //ボタン値という変数を定義する、ドライバ関数の key_values という配列と繋がる
    /*ボタンデバイス・オープン/dev/buttons*/
    buttons_fd = open("/dev/buttons", 0);
    if (buttons_fd < 0) {
        /*オープンが失敗な場合、終了*/
        perror("open device buttons");
        exit(1);
    }
    /*ボタンの値を読み取り、値と状態をプリントアウト*/
    for (;;) {
        char current_buttons[6];
        int count_of_changed_key;
        int i;
        /* read 関数で 1 セットのボタン値を読み取り（6 個）*/
        if (read(buttons_fd, current_buttons, sizeof current_buttons) != sizeof current_buttons) {
            perror("read buttons:");
            exit(1);
        }

        /*ボタン値を一つずつ分析する*/
        for (i = 0, count_of_changed_key = 0; i < sizeof buttons / sizeof buttons[0]; i++) {
            if (buttons[i] != current_buttons[i]) {
                buttons[i] = current_buttons[i];
                /*ボタン値をプリントアウト、ボタンの押し/離し状態を標識する/
                printf("%skey %d is %s", count_of_changed_key? ", ": "", i+1, buttons[i] == '0' ? "up" : "down");
                count_of_changed_key++;
            }
        }
        if (count_of_changed_key) {
            printf("¥n");
        }

    }
```

```
    /*ボタンデバイスファイル close*/
    close(buttons_fd);
    return 0;
}
```

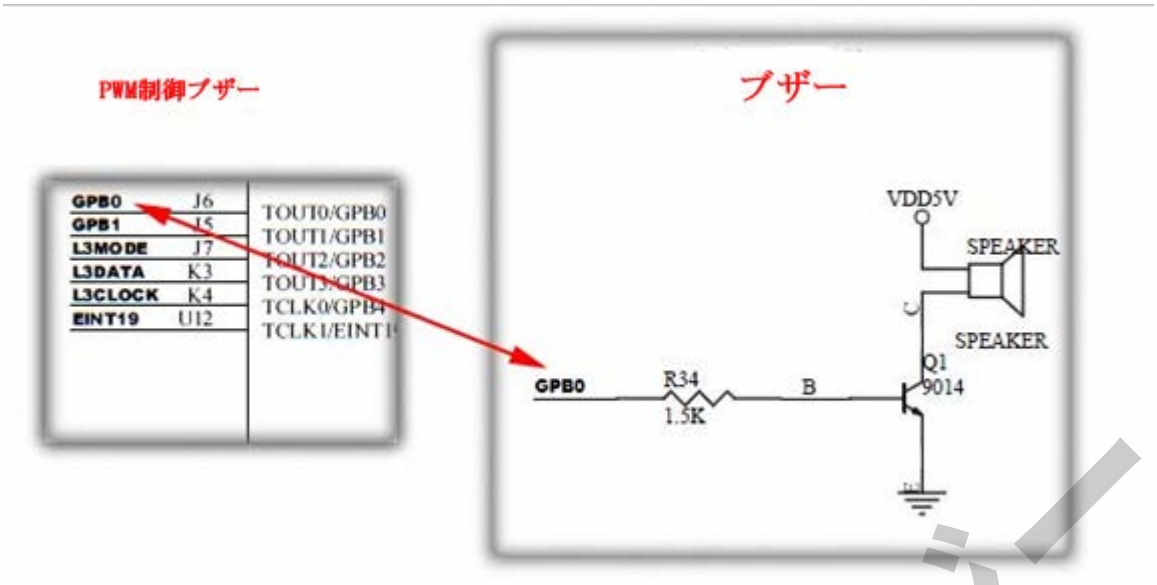
ソースコードのコマンドラインに：#arm-linux-gcc -o btn-test btn-test.c を実行し、btn-test という実行可能のバイナリファイルを生成する。USB メモリードライブや FTP ツールなどでファイルを開発ボードにコピーし、属性を実行権限に変更する。最後に実行し、ボタンテストを行う。下図を参照する：



3.24　ブザードライバを制御する PWM を追加

3.24.1 ハードウェア解析

Mini2440 ボードにブザーがあり、PWM で制御される。接続回路図は下図を参照する：



ブザーの GPB0 ポートの多重化機能は TOUT0、即ち PWM 出力。

S3C2440 データシートで下記記述がある：

TOUT0 is the PWM output. nTOUT0 is the inversion of the TOUT0. If the dead zone is enabled, the output wave form of TOUT0 and nTOUT0 will be TOUT0_DZ and nTOUT0_DZ, respectively. nTOUT0_DZ is routed to the TOUT1 pin.

In the dead zone interval, TOUT0_DZ and nTOUT0_DZ can never be turned on simultaneously.

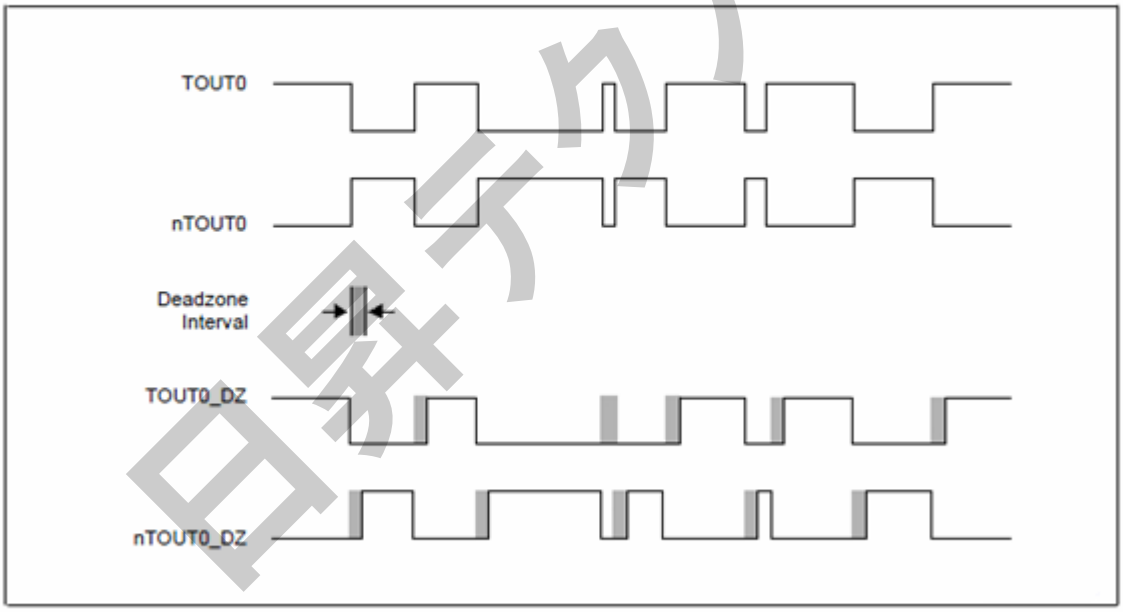


Figure 10-7. The Wave Form When a Dead Zone Feature is Enabled

ドライバに GPB0 ポートを PWM 機能出力に設定し、該当のタイマーを設定し、PWM の出力周波数を制御できる。

3.24.2 ドライバ追加

linux-2.6.32.2/drivers/char/ディレクトリ下にドライバファイル mini2440_pwm.c を追加する：

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/poll.h>
#include <linux/interrupt.h>
#include <linux/gpio.h>

#include <asm/irq.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <plat/regs-timer.h>
#include <mach/regs-irq.h>
#include <asm/mach/time.h>
#include <linux/clk.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/miscdevice.h>

#define DEVICE_NAME             "pwm" //デバイス名
#define PWM_IOCTL_SET_FREQ              1 //マクロ変数を定義、後の ioctl の switch case で使用する
#define PWM_IOCTL_STOP          0 //セマフォを定義 lock

static struct semaphore lock;

/* freq: pclk/50/16/65536 ~ pclk/50/16
 * if pclk = 50MHz, freq is 1Hz to 62500Hz
 * human ear : 20Hz~ 20000Hz
 */
static void PWM_Set_Freq( unsigned long freq ) //pwm 周波数設定、各レジスタコンフィグ
{
    unsigned long tcon;
    unsigned long tcnt;
    unsigned long tcfg1;
    unsigned long tcfg0;
```

```
    struct clk *clk_p;
    unsigned long pclk;

    //set GPB0 as tout0, pwm output     GPB0 を tout0,pwm 出力に設定
    s3c2410_gpio_cfgpin(S3C2410_GPB(0), S3C2410_GPB0_TOUT0);

    tcon = __raw_readl(S3C2410_TCON); //レジスタ TCON を tcon に読み取り
    tcfg1 = __raw_readl(S3C2410_TCFG1); //レジスタ TCFG1 を tcfg1 に読み取り
    tcfg0 = __raw_readl(S3C2410_TCFG0); //レジスタ TCFG0 を tcfg0 に読み取り

    //prescaler = 50
    tcfg0 &= ~S3C2410_TCFG_PRESCALER0_MASK; // S3C2410_TCFG_PRESCALER0_MASK タイマー0
と 1 のプリスケーラ値のマスクは TCFG[0~8]
    tcfg0 |= (50 - 1); // プリスケーラ 50

    //mux = 1/16
    tcfg1 &= ~S3C2410_TCFG1_MUX0_MASK; //S3C2410_TCFG1_MUX0_MASK タイマー 0 分割値のマ
スク：TCFG1[0~3]
    tcfg1 |= S3C2410_TCFG1_MUX0_DIV16; //タイマー 0 は 16 分割を行い

    __raw_writel(tcfg1, S3C2410_TCFG1); //tcfg1 の値を分割レジスタ S3C2410_TCFG1 に書き込み
    __raw_writel(tcfg0, S3C2410_TCFG0); //tcfg0 の値をプリスケール・レジスタ S3C2410_TCFG0 に書き込
み

    clk_p = clk_get(NULL, "pclk"); //pclk を得る
    pclk = clk_get_rate(clk_p);
    tcnt = (pclk/50/16)/freq; //タイマ入力クロックを取得後、PWM 変調周波数を設定

    __raw_writel(tcnt, S3C2410_TCNTB(0)); // PWM パルス幅変調周波数はタイマの入力クロックに等しい
    __raw_writel(tcnt/2, S3C2410_TCMPB(0)); //デューティ･サイクルは 50%

    tcon &= ~0x1f;

    tcon |= 0xb;  //disable deadzone, auto-reload, inv-off, update TCNTB0&TCMPB0, start timer 0
    __raw_writel(tcon, S3C2410_TCON); //tcon をカウンタ制御レジスタ S3C2410_TCON に書き込み
    tcon &= ~2;                 //clear manual update bit
    __raw_writel(tcon, S3C2410_TCON);
  }
```

```
static void PWM_Stop(void)
{
    s3c2410_gpio_cfgpin(S3C2410_GPB(0), S3C2410_GPIO_OUTPUT); // GPB0 を出力に設定
    s3c2410_gpio_setpin(S3C2410_GPB(0), 0); // GPB0 はロー・レベル、ブザー停止
}

static int s3c24xx_pwm_open(struct inode *inode, struct file *file)
{
    if (!down_trylock(&lock)) //セマフォ取得判断、取得 down_trylock(&lock) =0、未取得非 0
        return 0;
    else
        return -EBUSY; //エラー· メッセージを返します：要求されたリソースは利用できません
}

static int s3c24xx_pwm_close(struct inode *inode, struct file *file)
{
    PWM_Stop();
    up(&lock); //セマフォリリース lock
    return 0;
}

/*cmd   1、周波数設定；cmd   2 、pwm 停止*/
static int s3c24xx_pwm_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
    switch (cmd) {
    case PWM_IOCTL_SET_FREQ: //if cmd=1 、  case PWM_IOCTL_SET_FREQ に入ります
        if (arg == 0) //設定した周波数は 0
            return -EINVAL; //エラー· メッセージを返します、無効パラメータを伝送したという意味です。
        PWM_Set_Freq(arg); //or 周波数を設定
        break;
    case PWM_IOCTL_STOP: // if cmd=2 、  case PWM_IOCTL_STOP に入ります
        PWM_Stop(); //ブザー停止
        break;
    }
    return 0; //成功、戻り
}
```

```
/*デバイス･ファイル操作構造体を初期化*/
static struct file_operations dev_fops = {
    .owner    =   THIS_MODULE,
    .open     =   s3c24xx_pwm_open,
    .release  =   s3c24xx_pwm_close,
    .ioctl    =   s3c24xx_pwm_ioctl,
};

static struct miscdevice misc = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DEVICE_NAME,
    .fops = &dev_fops,
};

static int __init dev_init(void)
{
    int ret;

    init_MUTEX(&lock); // mutex を初期化
    ret = misc_register(&misc); //misc デバイス登録

    printk (DEVICE_NAME"¥tinitialized¥n");
        return ret;
}
static void __exit dev_exit(void)
{
    misc_deregister(&misc); //デバイスアンインストール
}
module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
MODULE_DESCRIPTION("S3C2410/S3C2440 PWM Driver");
```

上記ドライバプログラムに関する説明：

1 CPU カウンタ制御レジスタ

1）タイマ入力クロックを設定

TCFG0-クロック・コンフィギュレーション・レジスタ 0、プリスケーラ値（1~255）を取得する機能

TCFG1-クロック・コンフィギュレーション・レジスタ 1、分割値（2、4、8、16、32）を取得する機能

タイマ入力クロック周波数=PLCK/{プリスケーラ+1}/{分割値}

2）PWM のデューティサイクルを設定

TCNTB0-タイマー 0 カウント・バッファレジスタ、タイマ入力クロック周波数から得るパルス幅変調の周波数。TCMTB0-タイマー 0 比較バッファレジスタ、PWM のデューティサイクルを設定する機能、レジスタの値はハイー、例え TCNTB0 の周波数は 160、IF TCMTB0 は 110、 PWM は 110 目のサイクルはハイ、50 目サイクルはロー、したがって、デューティ･サイクルは 11：5

3）タイマコントロールレジスタ TCON

TCON[0~4]はタイマー 0 を制御する

2. レジスタ読み取り/書き込み関数：__raw_readl と__raw_writel

ポート・レジスタ読み取り関数 ：__raw_readl(a)、当関数はポート a から一つの 32 ビット値を返する。関連定義は include/asm-arm/io.h にある。#define __raw_readl(a) (*(volatile unsigned int*)(a))、ポート・レジスタ書き込み関数 ： __raw_writel(v、a)、当関数は一つの 32 ビット値をポート a に書き込み。関連定義は include/asm-arm/io.h にある。#define __raw_writel(v、a) (*(volatile unsigned int*)(a) = (v))。ここにファンクション･コントロール･レジスタを設定し、該当ピンを出力に設定する。

3. カーネルの GPIO 操作

gpio_cfgpin で対応 GPIO ポートコンフィグ

gpio_setpin IO ポート出力機能な場合、ピン書き込み

4. カーネルのセマフォに基づきの Linux の同時制御

ドライバでは、複数のスレッドが同時に同一のリソースにアクセスする時に、"レース"状態となり、共有リソースへの同時実行制御を行う必要がある。セマフォ（多くの場合はミューテックスとして使用される）は同時実行制御の手段として使用する（他にスピンロック、それは非常に短い期間を維持する場合に適している）。セマフォは、プロセスのコンテキストだけで使用されます。void init_MUTEX(&lock) mutex ロックを初期化する、すなわちセマフォを 1 に設定する

void up (&lock) セマフォリリース、待ち関数をウェイクアップ

int down_trylock(&lock) セマフォロック取得を試みる、セマフォ取得すると、0 を返する、or 非 0 を返する。そしてスリープを引き起こさず、割り込みコンテキストで使用できる。 PWM では、カウント値がオーバーフローすると、カウント割り込みが発生するため、この関数でセマフォ取得する。

3.24.3 ドライバをカーネルに追加

次に、linux-2.6.32.2/drivers/char/Kconfig ファイルをオープンし、赤字部分を追加する：

```
config MINI2440_BUTTONS
        tristate "Buttons driver for FriendlyARM Mini2440 development boards"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
        help
            this is buttons driver for FriendlyARM Mini2440 development boards

config MINI2440_BUZZER
        tristate "Buzzer driver for FriendlyARM Mini2440 development boards"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
        help
            this is buzzer driver for FriendlyARM Mini2440 development boards

config MINI2440_ADC
        bool "ADC driver for FriendlyARM Mini2440 development boards"
        depends on MACH_MINI2440
        default y if MACH_MINI2440
        help
            this is ADC driver for FriendlyARM Mini2440 development boards
            Notes: the touch-screen-driver required this option
```

linux-2.6.32.2/drivers/char/Makefile オープン、ドライバのターゲットファイルを構成定義に追加する、赤字コード：

```
obj-$(CONFIG_JS_RTC)                    += js-rtc.o
js-rtc-y = rtc.o

obj-$(CONFIG_LEDS_MINI2440)                     += mini2440_leds.o
obj-$(CONFIG_MINI2440_BUTTONS)      += mini2440_buttons.o
obj-$(CONFIG_MINI2440_BUZZER)       += mini2440_pwm.o
obj-$(CONFIG_MINI2440_ADC)          += mini2440_adc.o

# Files generated that shall be removed upon make clean
clean-files := consolemap_deftbl.c defkeymap.c
```

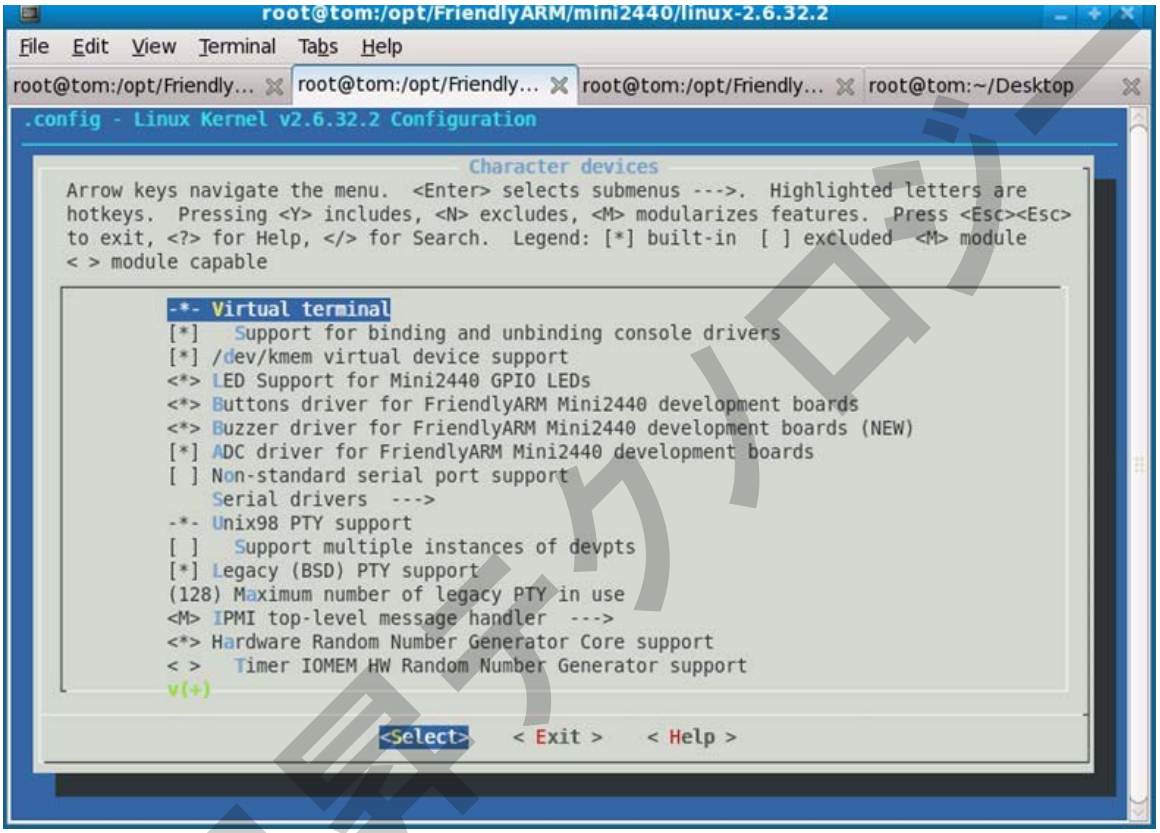
ここまで、カーネルにブザー制御の PWM ドライバを追加した。

3.24.4 新しいカーネルをコンフィグとコンパイル

上記の手順で、カーネルのソースコード・ディレクトリで下記のコマンドを実行する：make menuconfig カーネル再コンパイル、次のサブメニュに入る：

```
Device Drivers  --->
  Character devices  --->
```

PWM 制御ブザーのメニューに入る：



スペースキーで〈*〉Buzzer driver for FriendlyARM Mini2440 development boards(NEW)〉オプションを選択、上記のカーネル設定を保存し、終了する。

カーネルのソースコードのディレクトリで次のコードを実行：make zImage、生成された新カーネルを開発ボードにプログラミングする。

3.24.5 PWM 制御ブザーテスト

前章のボタンをテストするため、テストプログラムを作成する：pwm.c：

注：付属 DVD にテストプログラムのソースコードがある。位置は：¥linux¥examples¥pwm、ファイル名：pwm_test.c、ここでは名を pwm.c　に変更する。

```
#include <stdio.h> //標準入力と出力の定義
#include <termios.h> //POSIX 端末制御の定義
#include <unistd.h> //Unix 標準関数定義
#include <stdlib.h> //標準関数ライブラリ定義

#define PWM_IOCTL_SET_FREQ      1
#define PWM_IOCTL_STOP          0

#define ESC_KEY       0x1b // ESC の値は  ESC_KEY を定義される

static int getch(void) //端末で入力取得関数を定義し、入力値（int）を返します
{
struct termios oldt,newt; //端末構造 struct termios
int ch;

if (!isatty(STDIN_FILENO)) { //シリアルと標準入力の接続状態判断
fprintf(stderr, "this problem should be run at a terminal\n");
exit(1);
}
// save terminal setting
if(tcgetattr(STDIN_FILENO, &oldt) < 0) { //端末の設定パラメータを取得
perror("save the terminal setting");
exit(1);
}

// set terminal as need
newt = oldt;
newt.c_lflag &= ~( ICANON | ECHO ); //端末を制御し関数パラメータを編集； パラメータ ICANON は標準入力モード；パラメータ ECH0 は入力文字表示
if(tcsetattr(STDIN_FILENO,TCSANOW, &newt) < 0) { //新端末のパラメータを保存
perror("set terminal");
exit(1);
}

ch = getchar();

// restore termial setting
```

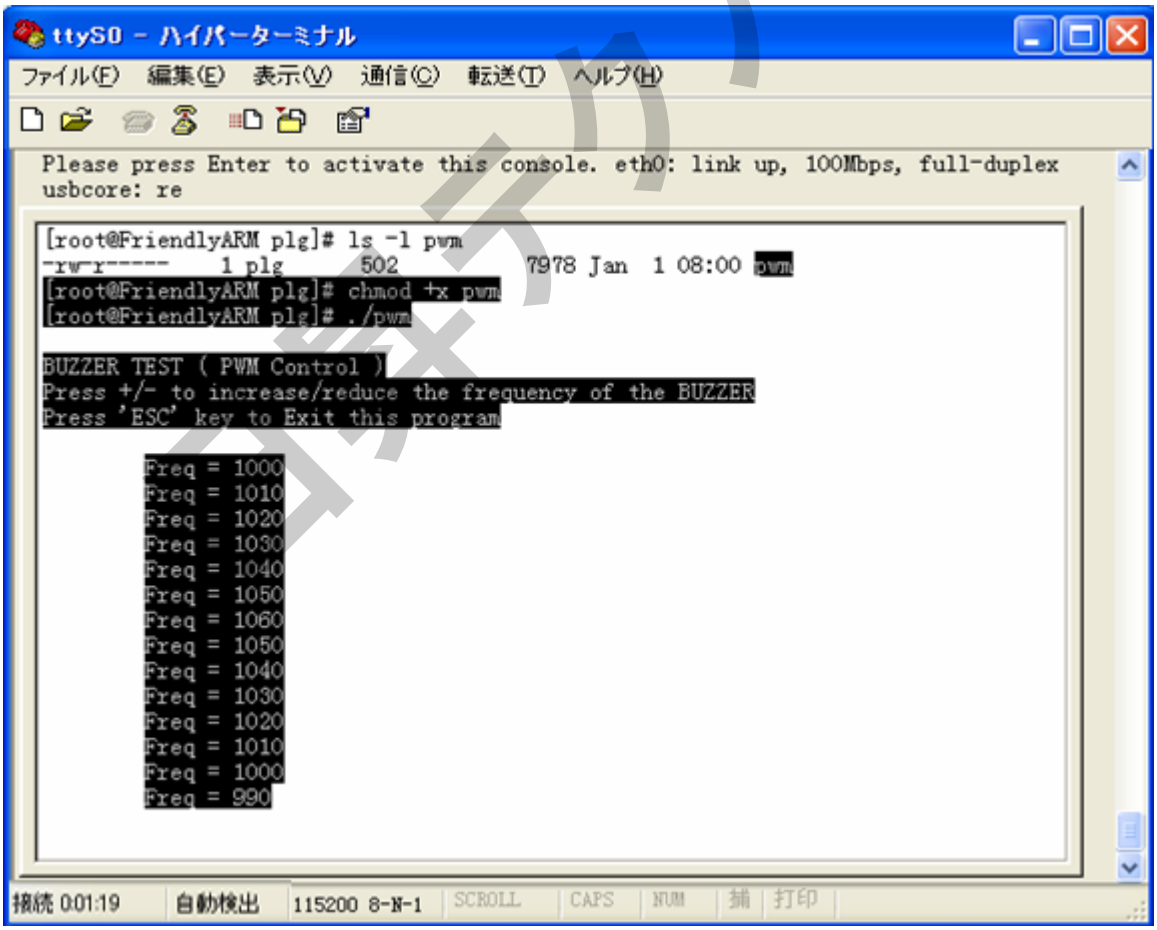
```
    if(tcsetattr(STDIN_FILENO,TCSANOW,&oldt) < 0) { //旧端末パラメータ復帰/保存
        perror("restore the termial setting");
        exit(1);
    }
    return ch;
}

static int fd = -1;
static void close_buzzer(void);
static void open_buzzer(void) //ブザーを開く
{
    fd = open("/dev/pwm", 0); // PWM のデバイス· ドライバ· ファイルを開く
    if (fd < 0) {
        perror("open pwm_buzzer device"); //開く際にエラーの場合、プロセス終了。終了パラメーターは 1
        exit(1);
    }

    // any function exit call will stop the buzzer
    atexit(close_buzzer); //コール・終了 close_buzzer
}

static void close_buzzer(void) //ブザー閉じる
{
    if (fd >= 0) {
        ioctl(fd, PWM_IOCTL_STOP); //ブザー閉じる
        close(fd); //デバイス・ドライバ・ファイルを閉じる
        fd = -1;
    }
}
static void set_buzzer_freq(int freq)
{
    // this IOCTL command is the key to set frequency
    int ret = ioctl(fd, PWM_IOCTL_SET_FREQ, freq); //周波数設定
    if(ret < 0) { //入力周波数誤差な場合
        perror("set the frequency of the buzzer");
        exit(1); //終了、　1 を返します
    }
}
```

```
static void stop_buzzer(void)
{
    int ret = ioctl(fd, PWM_IOCTL_STOP); //ブザー閉じる
    if(ret < 0) { //閉じない場合
        perror("stop the buzzer");
        exit(1); //終了、1 を返します
    }
}

int main(int argc, char **argv)
{
    int freq = 1000 ;

    open_buzzer();//ブザーを開く

    printf( "¥nBUZZER TEST ( PWM Control )¥n" );
    printf( "Press +/- to increase/reduce the frequency of the BUZZER¥n" ) ;
    printf( "Press 'ESC' key to Exit this program¥n¥n" );

    while( 1 )
    {
        int key;

        set_buzzer_freq(freq); //ブザー周波数を設定
        printf( "¥tFreq = %d¥n", freq );

        key = getch();

        switch(key) {
        case '+':
            if( freq < 20000 )
                freq += 10;
            break;

        case '-':
            if( freq > 11 )
                freq -= 10 ;
            break;
```

```
        case ESC_KEY:
        case EOF:
            stop_buzzer();
            exit(0);

        default:
            break;
        }
    }
}
```

pwm.c があるディレクトリからコマンドライン：#arm-linux-gcc -o pwm pwm.c を実行し、PWM 可実行バイナリ･ファイルを生成する。USB メモリードライブやFTP ツールなどでファイルを開発ボードにコピーして、属性を実行権限に変更する。最後に実行し、ボタンテストを行う。下図を参照する：

説明：テスト中ボタンボードのﾞ+ﾞ /ﾞ -ﾞ でPWM の出力周波数を調整できる。



第四章ファイルシステムについて

4.1 mini2440 root_qtopia ファイルシステムのスタートアッププロセスの分析

mini2440 最新の root_qtopia ファイルシステムのスタートアッププロセスについて、root_qtopia ファイルシステムの GUI は Qtopia に基づき、その初期化のスタートプロセスはほどんと busybox で完成する。Qtopia（qpe）はスタート最終段階だけで起動する。デフォルトカーネルコマンドに init=/linuxrc があるため、ファイルシステムがロードされた後、最初実行するプログラムはルートディレクトリ下の linuxrc である。 これは/bin/busybox へのポインタ・リンクで、即ち最初起動されたプログラムは busybox 自身である。

先ず、busybox は/etc/inittab を解析し、更なる初期設定情報を取得する。(busybox ソースコード init/init.c の parse_inittab()関数を参考ください)。次、root_qtopia に/etc/inittab というプロフィールが存在しないため、busybox のロジックにより、デフォルトのコンフィギュレーションを生成する：

コードコピー

```
1. static void parse_inittab(void)
2. {
3. #if ENABLE_FEATURE_USE_INITTAB
4. char *token[4];
5. parser_t *parser = config_open2("/etc/inittab", fopen_for_read);
6.
7. if (parser == NULL)
8. #endif
9. {
10. /* No inittab file -- set up some default behavior */
11. /* Reboot on Ctrl-Alt-Del */
12. new_init_action(CTRLALTDEL, "reboot", "");
13. /* Umount all filesystems on halt/reboot */
14. new_init_action(SHUTDOWN, "umount -a -r", "");
15. /* Swapoff on halt/reboot */
16. if (ENABLE_SWAPONOFF)
17. new_init_action(SHUTDOWN, "swapoff -a", "");
18. /* Prepare to restart init when a QUIT is received */
19. new_init_action(RESTART, "init", "");
20. /* Askfirst shell on tty1-4 */
21. new_init_action(ASKFIRST, bb_default_login_shell, "");
```

```
22. //TODO: VC_1 instead of ""? "" is console -> ctty problems -> angry users
23. new_init_action(ASKFIRST, bb_default_login_shell, VC_2);
24. new_init_action(ASKFIRST, bb_default_login_shell, VC_3);
25. new_init_action(ASKFIRST, bb_default_login_shell, VC_4);
26. /* sysinit */
27. new_init_action(SYSINIT, INIT_SCRIPT, "");
28. return;
29. }
```

一番大切なのは new_init_action(SYSINIT、 INIT_SCRIPT、 ″)、それは次の初期化スクリプトは INIT_SCRIPT が定義値で決める。マクロのデフォルト値は″/etc/init.d/rcS″。下記のはファイルシステム中

コードコピー

```
1. #! /bin/sh
2.
3. PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
4. runlevel=S
5. prevlevel=N
6. umask 022
7. export PATH runlevel prevlevel
8.
9. #
10. # Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
11. #
12. trap ":" INT QUIT TSTP
13. /bin/hostname FriendlyARM
14.
15. /bin/mount -n -t proc none /proc
16. /bin/mount -n -t sysfs none /sys
17. /bin/mount -n -t usbfs none /proc/bus/usb
18. /bin/mount -t ramfs none /dev
19.
20. echo /sbin/mdev > /proc/sys/kernel/hotplug
21. /sbin/mdev -s
22. /bin/hotplug
23. # mounting file system specified in /etc/fstab
24. mkdir -p /dev/pts
25. mkdir -p /dev/shm
```

```
26. /bin/mount -n -t devpts none /dev/pts -o mode=0622
27. /bin/mount -n -t tmpfs tmpfs /dev/shm
28. /bin/mount -n -t ramfs none /tmp
29. /bin/mount -n -t ramfs none /var
30. mkdir -p /var/empty
31. mkdir -p /var/log
32. mkdir -p /var/lock
33. mkdir -p /var/run
34. mkdir -p /var/tmp
35.
36. /sbin/hwclock -s
37.
38. syslogd
39. /etc/rc.d/init.d/netd start
40. echo " " > /dev/tty1
41. echo "Starting networking..." > /dev/tty1
42. sleep 1
43. /etc/rc.d/init.d/httpd start
44. echo " " > /dev/tty1
45. echo "Starting web server..." > /dev/tty1
46. sleep 1
47. /etc/rc.d/init.d/leds start
48. echo " " > /dev/tty1
49. echo "Starting leds service..." > /dev/tty1
50. echo " "
51. sleep 1
52.
53. /sbin/ifconfig lo 127.0.0.1
54. /etc/init.d/ifconfig-eth0
55.
56. /bin/qtopia &
57. echo " " > /dev/tty1
58. echo "Starting Qtopia, please waiting..." > /dev/tty1
```

/etc/init.d/rcS の内容は分析の重点でもある：

順次分析する：

```
1. PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
2. runlevel=S
3. prevlevel=N
4. umask 022
5. export PATH runlevel prevlevel
```

ブート環境に必要な環境変数を設定：

```
コードコピー
1. /bin/hostname FriendlyARM
```

マシン名を設定：

```
コードコピー
1. /bin/mount -n -t proc none /proc
2. /bin/mount -n -t sysfs none /sys
3. /bin/mount -n -t usbfs none /proc/bus/usb
4. /bin/mount -t ramfs none /dev
```

バーチャルファイルシステムをマウントし、｀/proc｀と｀/sys｀、そして/dev ディレクトリで ramfs をマウント、これは NAND Flash 上の読み取り専用の/dev ディレクトリ上に書き込み可能の SDRAM を上書きする。

次、/sys と ramfs をマウントした/dev は正確にデバイスノードを作成するキーポイントである。2.6.29 カーネルに対して、 devfs のサポートはないため、デバイスノードを作成する方法は 2 つある：

1）ファイルシステム・ミラーリングを作成する前に、mknod を使って手動でシステム内全てのデバイスノードを作成する、そしてデバイスノードをファイルシステムのミラーリングに加えます；

2）ファイルシステムの初期化処理中、/sys ディレクトリの出力情報により、/dev ディレクトリ下ダイナミックで目録システムの実際のデバイスノードを作成する。

方法 1）にはデバイスが動的に増減する場合に適して、多くのホットスワップデバイスがある場合には適用しない、例えばフラッシュドライブ、SD カードなど。

方法 2）は現在 PC Linux のやり方で（ udev に基づき）。2 つの前提が必要である：/ sys ディレクトリマウントと書き込み可能な/ dev ディレクトリ。

そして、システムファイル初期化する前に、/dev ディレクトリ下にはデバイスノート：/dev/console が必要である。

```
コードコピー
1. echo /sbin/mdev > /proc/sys/kernel/hotplug
2. /sbin/mdev -s
3. /bin/hotplug
```

1） mdev -s で/dev ディレクトリ内に必要なデバイスノードを作成する；

2）カーネルの hotplug handler を mdev に設定し、デバイスがホット･スワップすると、 mdev がカーネルからのメッセージを受信し該当の動作を行われる。例えば：USB メモリードライブ･マウント。

mdev について、ファイルシステムには/etc/mdev.conf ファイルが存在し、そして mdev の設定情報を含まれる。このファイルを介して、特定のニーズ用のデバイスノード名またはリンクをカスタムできる。下記は root qtopia の mdev.conf の内容：

コードコピー

```
1. # system all-writable devices
2. full 0:0 0666
3. null 0:0 0666
4. ptmx 0:0 0666
5. random 0:0 0666
6. tty 0:0 0666
7. zero 0:0 0666
8.
9. # console devices
10. tty[0-9]* 0:5 0660
11. vc/[0-9]* 0:5 0660
12.
13. # serial port devices
14. s3c2410_serial0 0:5 0666 =ttySAC0
15. s3c2410_serial1 0:5 0666 =ttySAC1
16. s3c2410_serial2 0:5 0666 =ttySAC2
17. s3c2410_serial3 0:5 0666 =ttySAC3
18.
19. # loop devices
20. loop[0-9]* 0:0 0660 =loop/
21.
22. # i2c devices
23. i2c-0 0:0 0666 =i2c/0
24. i2c-1 0:0 0666 =i2c/1
25.
26. # frame buffer devices
27. fb[0-9] 0:0 0666
28.
29. # input devices
30. mice 0:0 0660 =input/
31. mouse.* 0:0 0660 =input/
32. event.* 0:0 0660 =input/
33. ts.* 0:0 0660 =input/
34.
35. # rtc devices
```

```
36. rtc0 0:0 0644 >rtc
37. rtc[1-9] 0:0 0644
38.
39. # misc devices
40. mmcblk0p1 0:0 0600 =sdcard */bin/hotplug
41. sda1 0:0 0600 =udisk * /bin/hotplug
```

元シリアルドライバのデバイス登録名は s3c2410_serial0、 s3c2410_serial1 と s3c2410_serial2 で、mdev は/dev ディレクトリ下に ttySAC0、ttySAC1 と ttySAC2 を生成される。同じく、/dev/sdcard と /dev/udisk は SD カードおよび USB メモリードライブのファーストパーティションを指する

他の良く使われるファイルシステムをマウントする、そして/ var ディレクトリに（同じ RAMFS、書き込

```
コードコピー
1. # mounting file system specified in /etc/fstab
2. mkdir -p /dev/pts
3. mkdir -p /dev/shm
4. /bin/mount -n -t devpts none /dev/pts -o mode=0622
5. /bin/mount -n -t tmpfs tmpfs /dev/shm
6. /bin/mount -n -t ramfs none /tmp
7. /bin/mount -n -t ramfs none /var
8. mkdir -p /var/empty
9. mkdir -p /var/log
10. mkdir -p /var/lock
11. mkdir -p /var/run
12. mkdir -p /var/tmp
```

み可能）必要なディレクトリを作成する。

```
コードコピー
1. /sbin/hwclock -s
```

システム時刻を設定する機能、ハードウェア RTC から取得し、正しい時刻に設定する必要がある（RTC を設定する方法は、ユーザーマニュアルの説明を参照する）。現在開発ボードの時間はデフォルト時間である。次はシステムサービス起動で、log· レコード、サービスネットワーク、HTTP Server とカスタマイズされた″繰り返し点灯· サービス″を含む...

mdev.conf について：

```
コードコピー
1. # misc devices
2. mmcblk0p1 0:0 0600 =sdcard */bin/hotplug
3. sda1 0:0 0600 =udisk * /bin/hotplug
```

上記の機能は SD カードや USB メモリードライブの取り付け/取り外しの場合、情報をカスタマイズされたホットスワップ handler、 /bin/hotplug に伝送する。流れは、ファースト・パーティションを FAT/FAT32 として SD カードまたは USB メモリードライブにマウントする。

そして、フィールドテーブルが無いまたはファースト・パーティションが FAT/FAT32 ではない場合は適用しない。

/bin/hotplug のバイナリデータの一部は下記通り：

```
000010d0h: 52 00 00 00 4D 00 00 00 00 00 00 00 41 00 00 00 ; R...M.......A...
000010e0h: 43 00 00 00 54 00 00 00 49 00 00 00 4F 00 00 00 ; C...T...I...O...
000010f0h: 4E 00 00 00 00 00 00 00 44 00 00 00 45 00 00 00 ; N.......D...E...
00001100h: 56 00 00 00 4E 00 00 00 41 00 00 00 4D 00 00 00 ; V...N...A...M...
00001110h: 45 00 00 00 00 00 00 00 61 00 00 00 64 00 00 00 ; E.......a...d...
00001120h: 64 00 00 00 00 00 00 00 72 00 00 00 65 00 00 00 ; d.......r...e...
00001130h: 6D 00 00 00 6F 00 00 00 76 00 00 00 65 00 00 00 ; m...o...v...e...
00001140h: 00 00 00 00 2F 00 00 00 64 00 00 00 65 00 00 00 ; ..../...d...e...
00001150h: 76 00 00 00 2F 00 00 00 75 00 00 00 64 00 00 00 ; v.../...u...d...
00001160h: 69 00 00 00 73 00 00 00 6B 00 00 00 00 00 00 00 ; i...s...k.......
00001170h: 2F 00 00 00 64 00 00 00 65 00 00 00 76 00 00 00 ; /...d...e...v...
00001180h: 2F 00 00 00 73 00 00 00 64 00 00 00 63 00 00 00 ; /...s...d...c...
00001190h: 61 00 00 00 72 00 00 00 64 00 00 00 00 00 00 00 ; a...r...d.......
000011a0h: 4D 00 00 00 44 00 00 00 45 00 00 00 56 00 00 00 ; M...D...E...V...
000011b0h: 00 00 00 00 6D 00 00 00 6D 00 00 00 63 00 00 00 ; ....m...m...c...
000011c0h: 62 00 00 00 6C 00 00 00 6B 00 00 00 30 00 00 00 ; b...l...k...0...
000011d0h: 70 00 00 00 31 00 00 00 00 00 00 00 73 00 00 00 ; p...1.......s...
000011e0h: 64 00 00 00 61 00 00 00 31 00 00 00 00 00 00 00 ; d...a...1.......
000011f0h: 76 00 00 00 66 00 00 00 61 00 00 00 74 00 00 00 ; v...f...a...t...
00001200h: 00 00 00 00 2F 00 00 00 64 00 00 00 65 00 00 00 ; ..../...d...e...
00001210h: 76 00 00 00 2F 00 00 00 77 00 00 00 61 00 00 00 ; v.../...w...a...
00001220h: 74 00 00 00 63 00 00 00 68 00 00 00 64 00 00 00 ; t...c...h...d...
00001230h: 6F 00 00 00 67 00 00 00 00 00 00 00 9A B2 01 81 ; o...g........?
00001240h: B0 ; ?
```

```
コードコピー
1. syslogd
2. /etc/rc.d/init.d/netd start
3. echo ″                ″ > /dev/tty1
4. echo "Starting networking..." > /dev/tty1
5. sleep 1
6. /etc/rc.d/init.d/httpd start
7. echo "                              " > /dev/tty1
8. echo "Starting web server..." > /dev/tty1
9. sleep 1
10. /etc/rc.d/init.d/leds start
11. echo "                    " > /dev/tty1
12. echo "Starting leds service..." > /dev/tty1
13. echo "                    "
14. sleep 1
```

各サービスを起動:

syslog -カーネルとアプリケーションのデバッグ情報を記録

netd - inetd、各ネットワーク関連サービスをマウントするガードプロセス

httpd - http server ガードプロセス

leds -マーボタンガードプロセス

inetd 設定ファイル/etc/inetd.conf、内容は下記通り:

```
コードコピー
1. # /etc/inetd.conf: see inetd(8) for further informations.
2. echo          stream   tcp      nowait  root     internal
3. echo           dgram    udp     wait    root     internal
4. daytime       stream          tcp      nowait   root      internal
5. daytime       dgram   udp      wait     root     internal
6. time          stream   tcp      nowait   root     internal
7. time          dgram    udp      wait     root     internal
8.
9. # These are standard services.
10. #
11. ftp          stream   tcp      nowait   root /usr/sbin/ftpd          /usr/sbin/ftpd
12. telnet        stream          tcp      nowait   root /usr/sbin/telnetd       /usr/sbin/telnetd -i
```

起動したネットワークサービスは 2 つある： 1) ftp server と 2)telnet server。

```
コードコピー
1. /sbin/ifconfig lo 127.0.0.1
2. /etc/init.d/ifconfig-eth0
```

ネットワークデバイス（NIC）設定：

1）ローカル・ループバック・アドレスを 127.0.0.1 に設定する

2）NIC のセットアップスクリプトを実行する。 /etc/init.d/ifconfig-eth0

/etc/init.d/ifconfig-eth0 の内容は下記の通り。

```
コードコピー
1. #!/bin/sh
2.
3. echo -n Try to bring eth0 interface up.....>/dev/ttySAC0
4.
5. #/etc/eth0-setting ファイルの存在を判断します
6. if [ -f /etc/eth0-setting ] ; then
7.     #コンフィギュレーションファイルの情報を読み込み
8. source /etc/eth0-setting
9.
10.    #ルート･ファイル･システムは nfs の場合、NIC は既に設定したと意味し、再設定は必要ない
11. if grep -q "^/dev/root / nfs " /etc/mtab ; then
12. echo -n NFS root ... > /dev/ttySAC0
13.    #でないと、コンフィギュレーションファイル MAC, IP, Mask と Gateway に基づき、コマンド ifconfig で NIC を設定します。
14. else
15. ifconfig eth0 down
16. ifconfig eth0 hw ether $MAC
17. ifconfig eth0 $IP netmask $Mask up
18. route add default gw $Gateway
19. fi
20.
21.    #設定ファイル内の DNS を/etc/resolv.conf に書き込みます
22. echo nameserver $DNS > /etc/resolv.conf
23. #設定ファイルが存在しない場合、デフォルト設定を使用します
24. else
25.
26.    #ルート･ファイル･システムは nfs の場合、NIC は既に設定したと意味し、再設定は必要ない
27. if grep -q "^/dev/root / nfs " /etc/mtab ; then
28.    echo -n NFS root ... > /dev/ttySAC0
```

```
    else
            #ネットワークカードの IP アドレスを 192.168.1.230 に設定します
    /sbin/ifconfig eth0 192.168.1.230 netmask 255.255.255.0 up
    fi
fi

echo Done > /dev/ttySAC0
```

NFS の自動認識は/etc/mtab に NFS マウントされたかどうかのことにより実現する。

下記は root qtopia ファイルシステムの/etc/eth0-settings ファイル

```
コードコピー
IP=192.168.1.230
Mask=255.255.255.0
Gateway=192.168.1.1
DNS=192.168.1.1
MAC=08:90:90:90:90:90ySAC0
```

最後に、Qtopia GUI 起動

```
コードコピー
/bin/qtopia &
echo "                    " > /dev/tty1
echo "Starting Qtopia, please waiting..." > /dev/tty1
```

ここで、 Qtopia は/bin/qtopia を実行することで実現する。そして/bin/qtopia もスクリプトとなり、このスクリプトには Qtopia 実行の必要な環境を設定することである。最後に qpe という実行可能のファイルで Qtopia を起動する。内容は下記通り：

```
コードコピー
#!/bin/sh

#tslib 環境変数設定、 touchscreen デバイス・ファイル、tslib プロフィール、tslib plug-in 位置と touchscreen キャリブレーションデータファイル
export TSLIB_TSDEVICE=/dev/input/event0
export TSLIB_CONFFILE=/usr/local/etc/ts.conf
export TSLIB_PLUGINDIR=/usr/local/lib/ts
export TSLIB_CALIBFILE=/etc/pointercal
#Qtopia 環境変数設定、 Qtopia メインファイルの位置設定
export QTDIR=/opt/Qtopia
```

```
10. export QPEDIR=/opt/Qtopia
11. # PATH と LD_LIBRARY_PATH を設定します。 Qtopia の実行可能ファイルと共有ライブラリファイルをコンパイルして、Qtopia の環境を構成します。
12. export PATH=$QTDIR/bin:$PATH
13. export LD_LIBRARY_PATH=$QTDIR/lib:/usr/local/lib:$LD_LIBRARY_PATH
14.
15. # /sys/devices/virtual/input/input0/uevent を読み取り、touchscreen 情報があるかどうかを判断し、Qtopia が touchscreen と USB マウスの自動認識機能を実現します。
16. TS_INFO_FILE=/sys/devices/virtual/input/input0/uevent
17. if [ -e $TS_INFO_FILE -a "/bin/grep -q TouchScreen < $TS_INFO_FILE" ]; then
18. export QWS_MOUSE_PROTO="TPanel:/dev/input/event0 USB:/dev/input/mice"
19. if [ -e /etc/pointercal -a ! -s /etc/pointercal ] ; then
20. rm /etc/pointercal
21. fi
22. else
23. export QWS_MOUSE_PROTO="USB:/dev/input/mice"
24. >/etc/pointercal
25. fi
26. unset TS_INFO_FILE
27.
28. export QWS_KEYBOARD=TTY:/dev/tty1
29. export KDEDIR=/opt/kde
30.
31. export HOME=/root
32.
33. # /opt/Qtopia/bin/qpe を呼び出し Qtopia を起動します
34. exec $QPEDIR/bin/qpe 1>/dev/null 2>/dev/null
```

ここまで、ファイルシステムは初期化から Qtopia GUI 環境の最終起動までは全部終わる。

4.2 Busybox でファイル･システムを構築

4.2.1 busybox ソースコードをダウンロード

ダウンロード先：http://www.busybox.net/downloads/ 、バージョン：busybox-1.13.3.tar.gz、mini2440 開発ボードと同じのバージョン。

4.2.2 ルート･ファイル･システム･ディレクトリ・ヘルプ

組込み Linux はルートファイルシステムを構築する必要がある、ルートファイルシステム構築のルールは FHS(Filesystem Hierarchy Standard) ドキュメント内にある、下記はルートファイルシステムの最上位ディレクトリである。

ディレクト	内容

リ	
bin	全てのユーザーが使用できる基本コマンド
sbin	基本システムコマンド、システム起動、修復など
usr	共有、読み取り専用のプログラムやデータ
proc	空のディレクトリ、proc ファイル･システムのマウント･ポイントとして使用されます
dev	デバイス・ファイルやその他の特別なファイルが含まれています
etc	スタートアップファイルを含む、システムコンフィギュレーションファイル
lib	共用ライブラリーおよびロード可能なブロック（ドライバ）、共有ライブラリはシステム・ブート、ルート･ファイル･システムの実行可能プログラムに必要である
boot	ブートローダに使用の静的ファイル、
home	ユーザのホームディレクトリ、サービスアカウントのロックのホーム･ディレクトリを含む、例えば FTP
mnt	一時的にファイルシステムをマウントするために使用されるようなマウントポイント、通常は空のディレクトリ。中に空のサブディレクトリを作成する。
opt	ホスト追加のインストールソフトウェアのディレクトリ
root	root ユーザーのホームディレクトリ
tmp	一時ファイルの置き場所、通常は空ディレクトリ
var	変数データをの置き場

4.2.3 ルート･ファイル･システム･ディレクトリ作成

```
#!/bin/sh
echo "------Create rootfs directons start...--------"
mkdir rootfs
cd rootfs
echo "--------Create root,dev....----------"
mkdir root dev etc boot tmp var sys proc lib mnt home
mkdir etc/init.d etc/rc.d etc/sysconfig
mkdir usr/sbin usr/bin usr/lib usr/modules
echo "make node in dev/console dev/null"
mknod -m 600 dev/console c 5 1
mknod -m 600 dev/null c 1 3
mkdir mnt/etc mnt/jffs2 mnt/yaffs mnt/data mnt/temp
mkdir var/lib var/lock var/run var/tmp
chmod 1777 tmp
chmod 1777 var/tmp
echo "-------make direction done---------"
```

/opt/studyarm ディレクトリに入る、ルート･ファイル･システム･ディレクトリのスクリプトファイルを作成：create_rootfs_bash。コマンド chmod +x create_rootfs_bash でファイルの実行権限を変更する、./create_rootfs_bash スクリプトを実行し、ルート･ファイル･システム･ディレクトリの作成完了。

tmp ディレクトリ使用権が変更され、sticky ビットを開く、tmp ディレクトリの下に作成されたファイルを作成者だけから削除権限がある。組込みシステムにはほとんどシングルユーザーであるが、いくつかの組み込みアプリケーションは、必ずしも root 権限で実行しないため、従って、ルート･ファイル･システムのアクセス許可ビットは基本的な仕様を満たして設計する必要がある。

4.2.4 ダイナミック･リンク･ライブラリーの作成

ダイナミック･リンク･ライブラリーは付属 DVD 中のルート･ファイル･パッケージを使用する。lib の内容を作成したルート･ファイル･ディレクトリ lib にコピーする。

```
cd /mnt/hgfs/share
tar –zxvf root_qtopia.tgz –C /opt/studyarm
cp –rfd /opt/studyarm/root_qtopia/lib/* /opt/studyarm/rootfs/lib/*
```

4.2.5 Bosybox クロスコンパイラ

Bosybox は GPL v2 ライセンスに従うオープンソースプロジェクトで、開発中ファイルサイズを最適化し、そして限られたシステムリソース（メモリーなど）を配慮して、Busybox で自動的にルート･ファイル･システム必要な bin、sbin、usr ディレクトリと linuxrc ファイルを生成することができる

1、busybox 解凍

cd /mnt/hgfs/share

tar -zxvf busybox-1.13.3.tar.tgz -C /opt/studyarm

2、ソースコードに入り、Makefile を修正：

cd /opt/studyarm/busybox-1.13.3

修正：

CROSS_COMPILE ?=arm-linux- //第 164 行

ARCH ?=arm //第 189 行

3、busybox 設定

注：付属 DVD 中 busybox のソースコードパッケージは¥linux ディレクトリにある、ファイル名：busybox-1.13.3-mini2440.tgz、デフォルトのコンフィギュレーションファイルが含まれている：fa_config(コマンド゛ cp fa.config .config゛でコンフィグを使用可能で)。詳しい手順は：

make menuconfig でコンフィグする

(1)、Busybox Settings--->

General Configuration--->

[*] Show verbose applet usage messages

[*] Store applet usage messages in compressed form

[*] Support -install [-s] to install applet links at runtime

[*] Enable locale support(system needs locale for this to work)

[*] Support for -long-options
[*] Use the devpts filesystem for unix98 PTYs
[*] Support writing pidfiles
[*] Runtime SUID/SGID configuration via /etc/busybox.config
[*] Suppress warning message if /etc/busybox.conf is not readable
Build Options--->
[*] Build BusyBox as a static binary(no shared libs)
[*] Build with Large File Support(for accessing files>2GB)
Installation Options->
[]Don't use /usr
Applets links (as soft-links) --->
(./_install) BusyBox installation prefix
Busybox Library Tuning --->
(6)Minimum password legth
(2)MD5:Trade Bytes for Speed
[*]Fsater /proc scanning code(+100bytes)
[*]Command line editing
(1024)Maximum length of input
[*] vi-style line editing commands
(15) History size
[*] History saving
[*] Tab completion
[*]Fancy shell prompts
(4) Copy buffer size 、in kilobytes
[*]Use ioctl names rather than hex values in error messages
[*]Support infiniband HW

(2)、Linux Module Utilities--->
(/lib/modules)Default directory containing modules
(modules.dep)Default name of modules.dep
[*] insmod
[*] rmmod
[*] lsmod
[*] modprobe
-----options common to multiple modutils
[ ] support version 2.2/2.4 Linux kernels
[*]Support tainted module checking with new kernels
[*]Support for module .aliases file
[*] support for modules.symbols file

(3)、busybox で dev 下ディスクタイプのサポートコンフィギュレーション

dev の作成方法は 3 つある：

手動作成：ルート･ファイル･システムを作成するとき、dev ディレクトリ内に使用するデバイスファイルを作成する、システムがルートファイルシステムをマウントした後、dev ディレクトリのデバイスファイルを使用することができる。

devfs ファイルシステムを使用する：この方法ディスクマッピングは未定で、メジャー/マイナーデバイス番号の欠如、大量のメモリを消費する。現在この方法は古くなっています。

udev：ユーザ･プログラムである、システム･ハードウェアの状態に基づいて動的にディスク･ファイルを更新する。機能はディスクファイル削除、作成などで、。その操作は比較的複雑であるが、非常に高い柔軟性がある。

mdev は busybox 付属の簡易版 udev、組み込みアプリケーションに適し、使用は簡単である。機能はシステムは起動時に、ホット･スワップまたは動的にドライバをロードする時、自動的に必要なドライバノードファイルを生成することである。busybox に基づき組み込み Linux のルート･ファイル･システムを構築する時、それが一番良い選択である。

次のオプションは mdev のサポートを追加する。

```
Linux System Utilities --->
[*]Support /etc/mdev.conf
[*]Support command execution at device addition/removal
```

4、busybox コンパイル

busybox を指定したディレクトリにコンパイル：

```
cd /opt/studyarm/busybox-1.13.3
make CONFIG_PREFIX=/opt/studyarm/rootfs install
```

rootfs のディレクトリにディレクトリ bin、sbin、usr とファイル linuxrc を生成する。

4.2.6 etc ディレクトリ内の設定ファイル作成

1、etc/mdev.conf ファイル、中身は空。

2、/ etc ディレクトリにの passwd、group、shadow ファイルを rootfs の/ etc にコピーする

3、etc/sysconfig ディレクトリにファイル HOSTNAME を作成、内容は` H3-Studio` 。

4、etc/inittab 文件：

```
#etc/inittab
::sysinit:/etc/init.d/rcS
s3c2410_serial0::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```

5、etc/init.d/rcS 文件：

```
#!/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
runlevel=S
prevlevel=N
```

```
umask 022
export PATH runlevel prevlevel
echo "----------munt all---------------"
mount -a
echo /sbin/mdev>/proc/sys/kernel/hotplug
mdev -s
echo "***********************************************"
echo "***************Studying ARM********************"
echo "Kernel version:linux-2.6.29.1"
echo "Student:Huang huahai"
echo "Date:2009.10.1"
echo "***********************************************"
/bin/hostname -F /etc/sysconfig/HOSTNAME
(または/bin/hostname H3-Studio )
以下のコマンドを使って rcS の執行権限を変える：
Chmod +x rcS
6、etc/fstab ファイル：
#device mount-point type option dump fsck order
proc /proc proc defaults 0 0
none /tmp ramfs defaults 0 0
sysfs /sys sysfs defaults 0 0
mdev /dev ramfs defaults 0 0
7、 etc/profile ファイル：
#Ash profile
#vim:syntax=sh
#No core file by defaults
#ulimit -S -c 0>/dev/null 2>&1
USER="id -un"
LOGNAME=$USER
PS1='[¥u@¥h=W]#'
PATH=$PATH
HOSTNAME='/bin/hostname'
export USER LOGNAME PS1 PATH
```

4.2.7 ルート･ファイル･システム･イメージ･ファイル作成

下記のコマンドを使用し yaffs ファイルシステム作成ツールをインストールする：

```
cd /mnt/hgfs/share
tar –zxvf mkyaffs2image.tgz –C /
```

コマンド mkyaffs2image rootfs の rootfs.img を使用し、/opt/studyarm ディレクトリでルート·ファイル·システム·イメージ·ファイルを生成する。

スタートアップ情報：

Copy linux kernel from 0x00050000 to 0x30008000, size = 0x00200000 ... done

zImage magic = 0x016f2818

Setup linux parameters at 0x30000100

linux command line is: "noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0"

MACH_TYPE = 782

NOW, Booting Linux.....

Uncompressing Linux..... done, booting the kernel.

Linux version 2.6.29.1 (root@FriendlyARM) (gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)) #2 Thu Oct 1 16:46:24 JST 2009

CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177

CPU: VIVT data cache, VIVT instruction cache

Machine: Study-S3C2440

ATAG_INITRD is deprecated; please update your bootloader.

Memory policy: ECC disabled, Data cache writeback

CPU S3C2440A (id 0x32440001)

S3C24XX Clocks, (c) 2004 Simtec Electronics

S3C244X: core 405.000 MHz, memory 101.250 MHz, peripheral 50.625 MHz

CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on

Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256

Kernel command line: noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0

irq: clearing subpending status 00000003

irq: clearing subpending status 00000002

PID hash table entries: 256 (order: 8, 1024 bytes)

Console: colour dummy device 80x30

console [ttySAC0] enabled

Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)

Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)

Memory: 64MB = 64MB total

Memory: 60828KB available (3556K code, 302K data, 156K init)

Calibrating delay loop... 201.93 BogoMIPS (lpj=504832)

Mount-cache hash table entries: 512

CPU: Testing write buffer coherency: ok

net_namespace: 716 bytes

NET: Registered protocol family 16

S3C2410 Power Management, (c) 2004 Simtec Electronics

S3C2440: Initialising architecture

S3C2440: IRQ Support

S3C24XX DMA Driver, (c) 2003-2004,2006 Simtec Electronics

DMA channel 0 at c4808000, irq 33

DMA channel 1 at c4808040, irq 34

DMA channel 2 at c4808080, irq 35

DMA channel 3 at c48080c0, irq 36

S3C244X: Clock Support, DVS off

bio: create slab <bio-0> at 0

SCSI subsystem initialized

usbcore: registered new interface driver usbfs

usbcore: registered new interface driver hub

usbcore: registered new device driver usb

NET: Registered protocol family 2

IP route cache hash table entries: 1024 (order: 0, 4096 bytes)

TCP established hash table entries: 2048 (order: 2, 16384 bytes)

TCP bind hash table entries: 2048 (order: 1, 8192 bytes)

TCP: Hash tables configured (established 2048 bind 2048)

TCP reno registered

NET: Registered protocol family 1

NetWinder Floating Point Emulator V0.97 (extended precision)

JFFS2 version 2.2. (NAND) (SUMMARY) © 2001-2006 Red Hat, Inc.

yaffs Oct 1 2009 16:45:43 Installing.

msgmni has been set to 118

io scheduler noop registered

io scheduler anticipatory registered (default)

io scheduler deadline registered

io scheduler cfq registered

Console: switching to colour frame buffer device 30x40

fb0: s3c2410fb frame buffer device

lp: driver loaded but no devices found

ppdev: user-space parallel port driver

Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled

s3c2440-uart.0: s3c2410_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2440

s3c2440-uart.1: s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2440

s3c2440-uart.2: s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2440

brd: module loaded

loop: module loaded

dm9000 Ethernet Driver, V1.31

Uniform Multi-Platform E-IDE driver

ide-gd driver 1.18

ide-cd driver 5.00

Driver 'sd' needs updating - please use bus_type methods

S3C24XX NAND Driver, (c) 2004 Simtec Electronics

s3c2440-nand s3c2440-nand: Tacls=1, 9ns Twrph0=4 39ns, Twrph1=1 9ns

NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8-bit)

Scanning device for bad blocks

Bad eraseblock 3339 at 0x00000342c000

Creating 8 MTD partitions on "NAND 64MiB 3,3V 8-bit":

0x000000000000-0x000000030000 : "bootloder"

0x000000050000-0x000000250000 : "kernel"

0x000000250000-0x000003ffc000 : "root"

0x000000800000-0x000000a00000 : "S3C2410 flash partition 3"

0x000000a00000-0x000000e00000 : "S3C2410 flash partition 4"

0x000000e00000-0x000001800000 : "S3C2410 flash partition 5"

0x000001800000-0x000003000000 : "S3C2410 flash partition 6"

0x000003000000-0x000004000000 : "S3C2410 flash partition 7"

usbmon: debugfs is not available

ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver

s3c2410-ohci s3c2410-ohci: S3C24XX OHCI

s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1

s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000

usb usb1: configuration #1 chosen from 1 choice

hub 1-0:1.0: USB hub found

hub 1-0:1.0: 2 ports detected

usbcore: registered new interface driver libusual

usbcore: registered new interface driver usbserial

USB Serial support registered for generic

usbcore: registered new interface driver usbserial_generic

usbserial: USB Serial Driver core

USB Serial support registered for FTDI USB Serial Device

usbcore: registered new interface driver ftdi_sio

ftdi_sio: v1.4.3:USB FTDI Serial Converters Driver

USB Serial support registered for pl2303

usbcore: registered new interface driver pl2303

pl2303: Prolific PL2303 USB to serial adaptor driver

mice: PS/2 mouse device common for all mice

S3C24XX RTC, (c) 2004,2006 Simtec Electronics

s3c2440-i2c s3c2440-i2c: slave address 0x10

s3c2440-i2c s3c2440-i2c: bus frequency set to 98 KHz

s3c2440-i2c s3c2440-i2c: i2c-0: S3C I2C adapter

S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics

```
s3c2410-wdt s3c2410-wdt: watchdog inactive, reset disabled, irq enabled
TCP cubic registered
NET: Registered protocol family 17
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 3191 is marked bad
block 3192 is bad
yaffs_read_super: isCheckpointed 0
VFS: Mounted root (yaffs filesystem) on device 31:2.
Freeing init memory: 156K
------mount all------
**********************************
**********studyARM*******************
Kernel version:linux-2.6.29.1
Student:Huang huahai
Data:2009.10.1
**********************************
Please press Enter to activate this console.
```

4.3 mdev の使用方法および原理

mdev は busybox 付属の udev の簡易版であり、組み込みアプリケーションに適する。機能は使用の際に簡単である。その機能は、システム起動時およびホット･スワップまたは動的にドライバをロードしている時に、自動的にドライバーが必要とするノードのファイルを生成する。 busybox に基づいて組込み Linux のルートファイルシステムを構築する時には一番良い選択である。

4.3.1 mdev 使用

mdev の使用は busybox の mdev.txt ドキュメント内には詳しい説明：

(1) コンパイル時 MDEV のサポートを追加

説明: busybox1.10.1 を使用

Linux System Utilities ---> mdev

Support /etc/mdev.conf

Support command execution at device addition/removal

(2) スタートの時 mdev を使用するコマンドを追加する

作成のルートファイルシステムに（nfs）の/linuxrc ファイルに下記のコマンドを追加する:

#/sys を sysfs ファイルシステムにマウント

```
    echo "----------mount /sys as sysfs"
    /bin/mount -t tmpfs mdev /dev
    /bin/mount -t sysfs sysfs /sys
    echo "----------Starting mdev......"
    /bin/echo /sbin/mdev > /proc/sys/kernel/hotplug
    mdev -s
```

注：/bin/echo /sbin/mdev > /proc/sys/kernel/hotplug

busybox ドキュメントエラー！！

(3) ドライバにクラスのデバイスインタフェースサポートを追加する

ドライバーの中初期化関数中、クラスデバイスディレクトリ下にデバイス番号を含むプロパティファイルを追加する、ファイル名は"dev"である。そして mdev で /dev ディレクトリにデバイスノードファイル gpio_dev0 を生成する。

```
    my_class = class_create(THIS_MODULE、 "gpio_class");
    if(IS_ERR(my_class)) {
        printk("Err: failed in creating class.¥n");
        return -1;
    }
    /* register your own device in sysfs、 and this will cause mdev to create corresponding
device node */
    class_device_create(my_class、 MKDEV(gpio_major_number、 0)、 NULL、
"gpio_dev%d" 、0);
```

ドライバーの中でクリアブロックで次のステートメントを追加し、クリーンアップを完了する。

```
    class_device_destroy(my_class、 MKDEV(gpio_major_number、 0));
    class_destroy(my_class);
```

必要なヘッダファイルは linux/device.h、プログラムの先頭には、次の内容を追加する必要がある：

```
#include <linux/device.h>
```

一方、my_class は class 型の構造体へのポインタ、プログラムの先頭でグローバル変数に宣言する必要がある。

```
    struct class *my_class;
```

上記コマンドに gpio_major_number は デバイスのメインノード番号である。必要なノード番号に変更できる。

gpio_dev は最終のデバイスノードファイル名である。%d は同じデバイスを自動ナンバリング機能である。gpio_class は class の名称を生成する、ドライバがロードされたとき、/sys/class ディレクトリに確認できる。

上記のステートメントは、必ずしも、初期化とクリーンアップフェーズで使用するわけではなく、別の場合でも使用できる。

(4)/etc/mdev.conf 文件

/etc/mdev.conf がなくても使用上に支障はありません。

4.3.2 mdev 原理

(1)mdev -s 実行

説明：‘-s’をパラメータとして、/sbin ディレクトリの mdev を呼び出し（実はリンクで、機能は/ bin ディレクトリの下に busybox のプログラムにパラメータを渡して呼び出すこと)、mdev は/sys/class と /sys/block の中すべてのクラスのデバイスカタログをスキャンし、ディレクトリに ″dev″という名前のファイルが含まれている場合、そしてファイルは、デバイス番号が含まれている場合、 mdev はこの情報を使用してこのデバイス用の/ dev ファイルにデバイスノードを作成する。一般的に起動する時一度だけで` mdev -s` を起動する。

(2) ホットプラグイベント

起動時にコマンドを実行：echo /sbin/mdev > /proc/sys/kernel/hotplug 、ホットプラグイベントを発生する時、カーネルは/ sbin ディレクトリの mdev を呼び出する。mdev は環境変数 ACTION と DEVPATH を通じて、（システム・デフォルト・アブリ）今回の hotplug イベントが/ sys のどのディレクトリに影響したのを判断する。次にこのディレクトリ内 ″dev″という属性ファイルがあるかどうかを確認し、あればこのデバイスの/dev ファイルにデバイスノードを作成する。

4.3.3 mdev で gpio 制御サンプル

サンプル内容は下記通り。

```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/init.h>
#include <linux/kernel.h>       /* printk() */
#include <linux/fs.h>           /* everything... */
#include <linux/cdev.h>
#include <linux/interrupt.h>    /* request_irq() */
#include <asm/arch/regs-gpio.h>
#include <asm/arch/regs-irq.h>
#include <asm/io.h>
#include <asm/uaccess.h>            /* copy_to_user() */
#include <linux/delay.h>            /* mdelay() */
#include <linux/device.h>       /*class_create()*/
#include <asm/arch-s3c2410/regs-gpio.h>
#include <asm/arch-s3c2410/regs-timer.h>

#define VERSION_STRING "gpio driver for JM_Xcontrol"

#define DEVICE_NAME "JM_Xcontrol_gpio"

/* Use 0xE0 as magic number */
#define XRAY_IOC_MAGIC 0xE0
```

```
    else
    {
        s3c2410_gpio_setpin(S3C2410_GPB1, 0);
    }
    break;
  case XRAY_IOC485REC:
    if(arg==0)
    {
        s3c2410_gpio_setpin(S3C2410_GPG10, 1);
    }
    else
    {
        s3c2410_gpio_setpin(S3C2410_GPG10, 0);
    }
    break;
  case XRAY_IOC485TRC:
    if(arg==0)
    {
        s3c2410_gpio_setpin(S3C2410_GPG12, 0);
    }
    else
    {
        s3c2410_gpio_setpin(S3C2410_GPG12, 1);
    }
    break;
  case XRAY_IOCBUZZER:
    if(arg==0)
    {
        s3c2410_gpio_setpin(S3C2410_GPB0, 0);
    }
    else
    {
        s3c2410_gpio_setpin(S3C2410_GPB0, 1);
    }
    break;
  default:
    break;
    }
    return 0;
}
```

```
static struct file_operations gpio_fops = {
 .owner = THIS_MODULE,
 //.open = xray_open,
 //.release = xray_release,
 //.read = xray_read,
 //.write = xray_write,
 .ioctl = gpio_ioctl,
 //.fasync = xray_fasync,
};

static int __init gpio_init(void)
{
 int ret,devno;
 dev_t dev;
 unsigned long tmp;
 ret = alloc_chrdev_region(&dev,0,1,DEVICE_NAME);
 gpio_major_number = MAJOR(dev);
 printk(KERN_INFO "Initial jm_xcontrol_gpio driver!¥n");
 if (ret<0) {
     printk(KERN_WARNING "gpio:can't get major number %d¥n",gpio_major_number);
     return ret;
 }

 devno = MKDEV(gpio_major_number,0);
 cdev_init(&gpio_dev,&gpio_fops);
 gpio_dev.owner = THIS_MODULE;
 gpio_dev.ops = &gpio_fops;

 ret = cdev_add(&gpio_dev,devno,1);
 if (ret) {
     unregister_chrdev_region(dev,1);
     printk(KERN_NOTICE "Error %d adding gpio device¥n",ret);
     return ret;
 }
 my_class = class_create(THIS_MODULE, "gpio_class");
 if(IS_ERR(my_class)) {
     printk("Err: failed in creating class.¥n");
     return -1;
 }
```

```
 /* register your own device in sysfs, and this will cause mdev to create corresponding device node */
 class_device_create(my_class, MKDEV(gpio_major_number, 0), NULL, "gpio_dev%d" ,0);

 //LCD バックライト
 s3c2410_gpio_cfgpin(S3C2410_GPB1, S3C2410_GPB1_OUTP);
 s3c2410_gpio_setpin(S3C2410_GPB1, 0);
 //ブザー
 s3c2410_gpio_cfgpin(S3C2410_GPB0, S3C2410_GPB0_OUTP);
 s3c2410_gpio_setpin(S3C2410_GPB0, 0);

 //485 トランシーバ制御
 s3c2410_gpio_cfgpin(S3C2410_GPG10, S3C2410_GPG10_OUTP); //收
 s3c2410_gpio_setpin(S3C2410_GPG10, 0);
 s3c2410_gpio_cfgpin(S3C2410_GPG12, S3C2410_GPG12_OUTP); //发
 s3c2410_gpio_setpin(S3C2410_GPG12, 0);
 return 0;
}

static void __exit gpio_cleanup(void)
{
 unsigned long tmp;
 dev_t dev=MKDEV(gpio_major_number,0);
 cdev_del(&gpio_dev);
 class_device_destroy(my_class, MKDEV(gpio_major_number, 0));
 class_destroy(my_class);
 unregister_chrdev_region(dev,1);
 s3c2410_gpio_setpin(S3C2410_GPB1, 1); //バックライトオフ
 s3c2410_gpio_setpin(S3C2410_GPB0, 0); //ブザーオフ

 s3c2410_gpio_setpin(S3C2410_GPG10, 1); //485 受信オフ
 s3c2410_gpio_setpin(S3C2410_GPG12, 0); //485 送信オフ
 printk(KERN_INFO "unregistered the %s¥n",DEVICE_NAME);
}

 module_init(gpio_init);
 module_exit(gpio_cleanup);
```