

Multi-Media ARM9

QQ2440v3 Linux 版

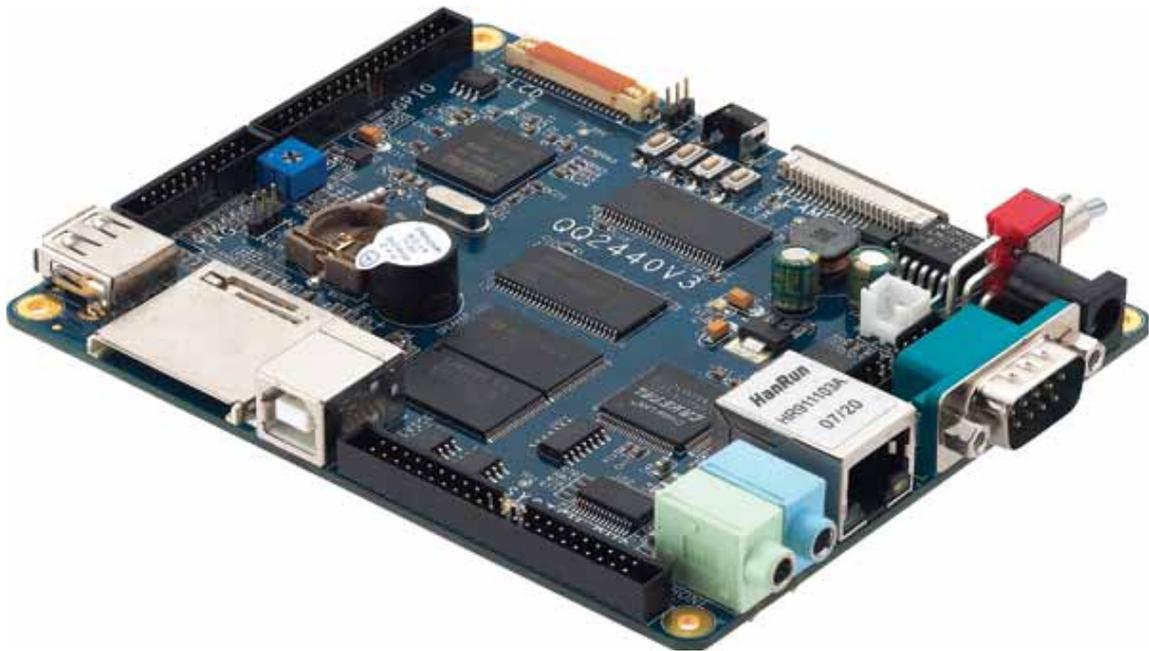
マニュアル

株式会社日昇テクノロジー

<http://www.csun.co.jp>

info@csun.co.jp

2009/2/2



[copyright@2008-2009](http://www.csun.co.jp)



第一章 QQ2440v3 ボードの概要	5
1.1 仕様	5
1.2 使えるデバイス例	8
1.3 付属アプリケーション例	9
第二章 QQ2440v3 ボードの初体験	10
2.1 パソコン側のハイパーターミナルの設定	10
2.2 ハイパーターミナルのコンソールで遊ぶ	12
2.2.1 MP3 をディスプレイしよう	12
2.2.2 USBメモリの使用	12
2.2.3 SD/MMCメモリの使用	12
2.2.4 USBカメラで撮影した画像を取る	13
2.2.5 ボタンのテスト	13
2.2.6 シリアルポート 1/2 のテスト	13
2.2.7 ブザーのテスト	13
2.2.8 液晶のバックライトの制御	13
2.2.9 I2Cデバイス(24C04)のテスト	13
2.2.10 RTCの設定	14
2.2.11 液晶(LCD)画面を取ります	14
2.3 QQ2440v3 のネットワーク機能	14
2.3.1 QQ2440v3 のウェブサーバー	14
2.3.2 QQ2440v3 のTelnetとFtp機能	14
2.3.3 DNSとgatewayの設定	14
2.3.4 MACアドレスの設定	14
2.3.5 ネットワーク・ファイルシステム(NFS)のマウント	15
第三章 Linuxのクロス開発環境	16
3.1 クロス開発環境を構築	16
3.2 NFSサーバーを構築	17
3.3 NFSはルートファイルシステムとして起動	17
第四章 Linuxのアプリケーションを開発	18
4.1 Hello, World!	18
4.2 Hello,Worldをコンパイル	18
4.2 Hello,WorldをARM9 ボードで実行	18
4.3 他のサンプル	18
第五章 Linuxカーネルを再構築	20
5.1 カーネルのソースコードを解凍	20



5.2	コンフィグ	20
5.3	カーネルをコンパイル	22
5.4	ドライバの場所	22
5.5	デバイスドライバのコンフィグ	23
5.5.1	LCDのコンフィグ	23
5.5.2	タッチパネルのコンフィグ	26
5.5.3	USBマウスとキーボード	28
5.5.4	USBメモリ	30
5.5.5	USBカメラ	32
5.5.6	CS8900 Ethernet	35
5.5.7	オーディオ	39
5.5.8	SD/MMC	42
5.5.9	LED	42
5.5.10	ボタン	43
5.5.11	シリアルポート	44
5.5.12	RTC	45
5.5.13	yaffsファイルシステム	46
5.5.14	EXT2/VFAT/ NFSファイルシステム	48
5.6	yaffsルートファイルシステムのイメージを生成	52
5.7	Linuxドライバの開発入門	53
5.7.1	簡単なドライバのソースコード	53
5.7.2	コンフィグファイルを編集します	54
5.7.3	Makefileを編集	56
5.7.4	ドライバをコンパイルします	56
5.7.5	ARM9 ボードでドライバをインストールします	57
第六章	生成されたファイルを書き込む	59
6.1	NOR Flashから起動	59
6.2	USBドライバのインストール	59
6.3	NAND Flashのパーティション	62
6.4	ブートロードの書き込み	63
6.5	Linuxのカーネルの書き込み	66
6.6	ルート・ファイルシステムの書き込み	67
6.7	NAND Flashのバックアップ	68
6.8	NAND Flashのリストア	70
6.9	メモリでLinuxカーネルを直接に実行	72
第七章	NOR Flashのブートロードを更新	75

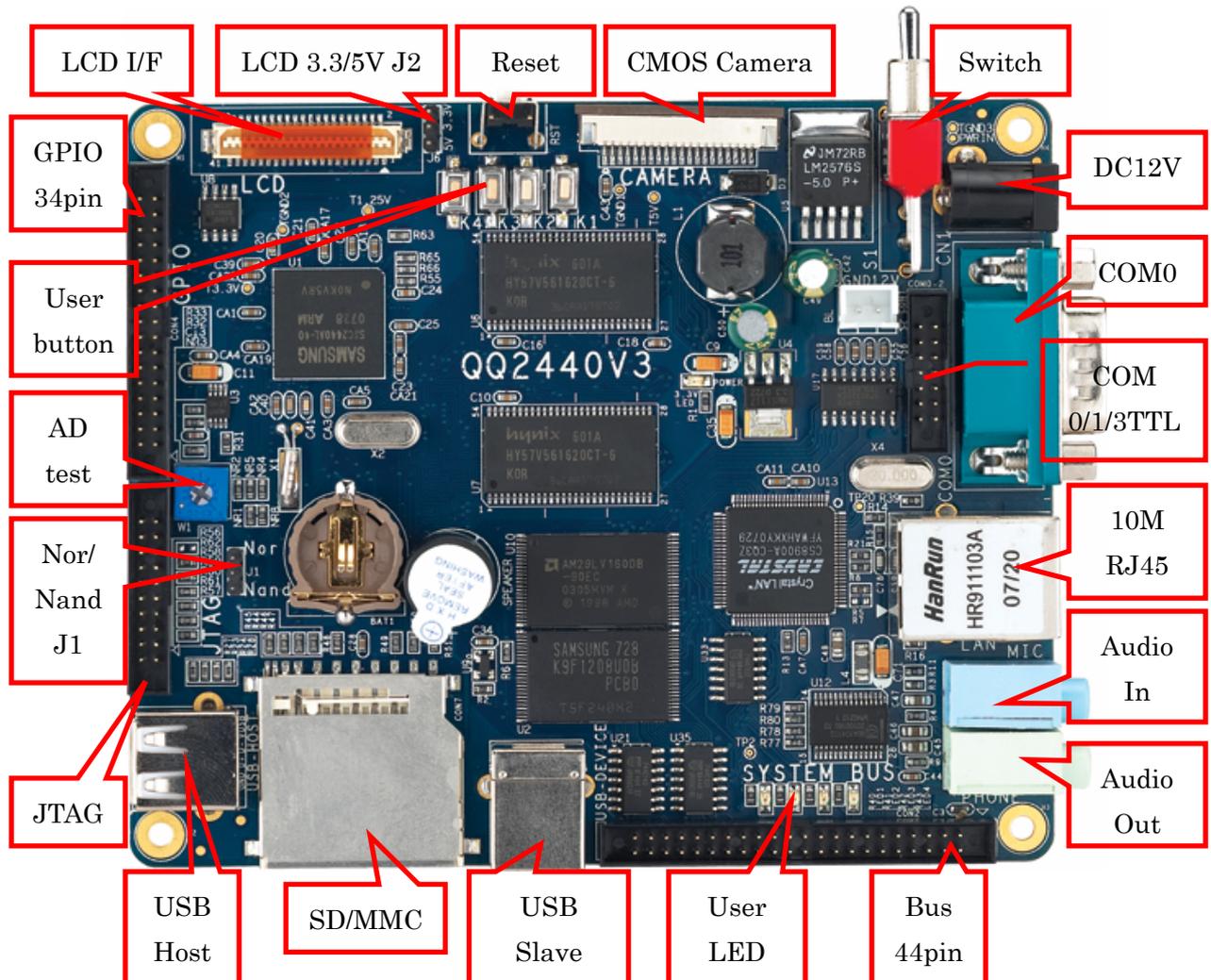


7.1 H-JTAGをダウンロードとインストールします	75
7.2 NOR Flashを書き込む	79
付録 1 Qt/Embedded GUIプログラムを作る	83
付録 2 アプリケーションを移植	90
付録 3 Watchdogの使い方	95
付録 4 ADの使い方	96
付録 5 UVC(USB Device Class)Webカメラを使用	97
付録 6 Mini GUIの移植とプログラム設計	99
付録 7 ADS1.2 とH-JTAGでプログラムを開発	109
付録 8 U-boot	120

使用されたソースコードは<http://www.csun.co.jp/>からダウンロードできます。

第一章 QQ2440v3 ボードの概要

1.1 仕様



液晶は3.3V/5V二種類がありますQQ2440v3の電圧選択ジャンパー(J2)は必ず正しく設定されなければなりません。

CPU プロセッサー

- ARM920T コアを採用したサムソン(SAMSUNG)社の S3C2440A、周波数 400MHz、最高周波数 533MHz。



メモリ

- 64MB SDRAM, 32 ビット幅データ・バス, SDRAM の最高周波数 100MHz
- 64MB NAND Flash メモリ
- 2MB NOR Flash メモリ

液晶(LCD)

- 4 線式抵抗膜方式のタッチパネルのインターフェース
- 標準の LCD I/F を持って、3.5”から 12.1”までの各種液晶パネル(黒白、STN、TFT、最高分解能 1024*768)に対応します。

インターフェース

- 10MBase-T Ethernet RJ45(CS8900) × 1
- RS232 シリアルポート × 3
- USB1.1 ホスト × 1
- USB1.1 スレーブ × 1
- MMC/SD メモリカードのソケット × 1
- ステレオ・オーディオの出力 × 1
- マイクの入力 × 1
- 10 ピンの JTAG(2mm DIP ピッチ)
- ユーザー LED × 4
- ユーザー ボタン × 4
- PWM 制御の圧電ブザー × 1
- 可変抵抗、A/D のテストの為に × 1
- I2C バスの AT24C08、I2C バスのテストの為に × 1
- 20 ピン CMOS カメラのインターフェース(2mm DIP ピッチ)
- RTC のバッテリーバックアップ
- 34 ピン GPIO(2mm DIP ピッチ)
- 44 ピンのシステムバス(2mm DIP ピッチ)

搭載した OS

- Linux2.6.13
- WindowsCE.NET 5.0
- uCOSII

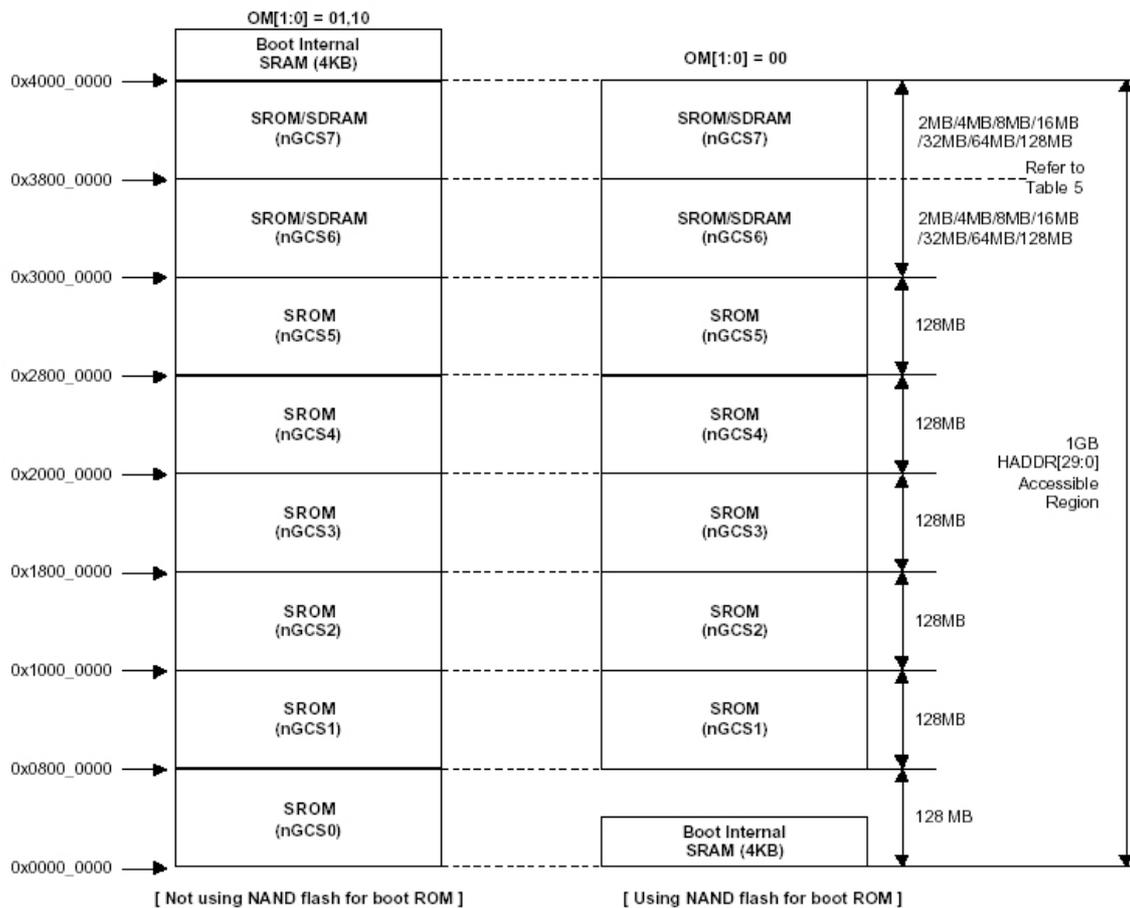
外形寸法

- 120×100(mm) 突起物は除く

供給電源

- 12VDC 電源、プラグ 2.1mmφ、極性はセンタープラス  です。電源スイッチと電源指示 LED 付き

ジャンパー-J2 はボードの動作モデルを選択します。一つは Nor Flash から起動です。もう一つは Nand Flash から起動です。この二つの起動モデルでシステムのアドレスが異なります。



デフォルトの設定は Nand Flash から Linux を起動します。

1.2 使えるデバイス例



USB カメラ
(SPACXX 又は
UVC に対応)



USB 無線 LAN 装置



USB マウスとキーボード



外付けハードディスク

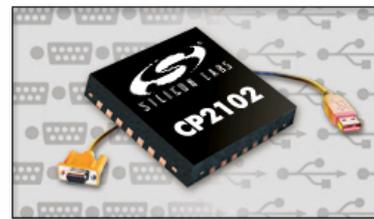
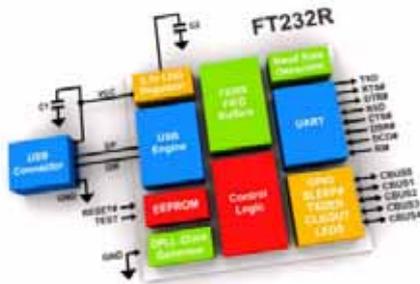


2GB までの
SD/MMC メモリ



USB HUB

USB メモリ



USB シリアルポート



液晶

付属のドライバ以外は、使えない可能性があります。

1.3 付属アプリケーション例



Qtopia デスク
トップ

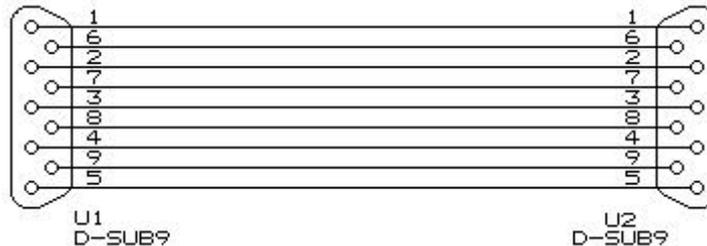
mpeg 映画

MP3
displayer

Web
ブラウザー

第二章 QQ2440v3 ボードの初体験

- DB9 メス-メス型のストレートケーブルで QQ2240v3 とパソコンを繋ぐ。



- クロス LAN ケーブルで QQ2240v3 とパソコンを繋ぐ。
- QQ2240v3 のオーディオ出力とスピーカーを繋ぐ。
- 12V 電源、極性はセンタープラス $\ominus - \oplus$ です。電源を入れると：



Linux の起動画面

パソコンがなくでも遊べます。

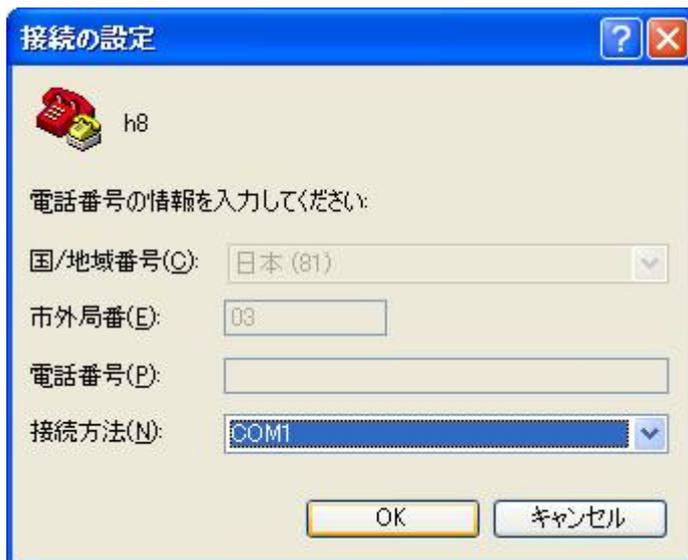


2.1 パソコン側のハイパーターミナルの設定

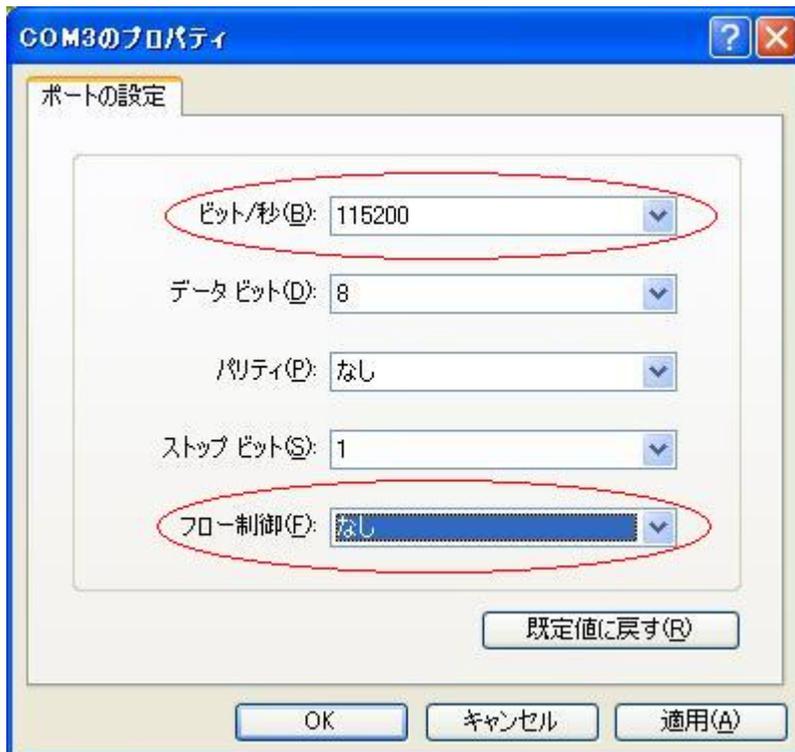
パソコンのメニュー：スタート → すべてのプログラム → アクセサリ → 通信 → ハイパーターミナルを選ぶと、次の画面が出てきます。



このハイパーターミナルの名前を入力して、"OK"ボタンを押すと。



使用しているシリアルポートを選んでください。



シリアル通信速度を 115200bps に設定します。フロー制御はなしです。
"OK"ボタンを押すと、設定が完了します。

2.2 ハイパーターミナルのコンソールで遊ぶ

2.2.1 MP3 をディスプレイしよう

```
# madplay your.mp3
```

madplay -h は madplay のヘルプを表示します。

2.2.2 USB メモリの使用

USB メモリを USB ホスト又は USB ハブに挿入して、次のコマンド：

```
# mount /dev/sda1 /mnt
```

```
# ls /mnt
```

2.2.3 SD/MMC メモリの使用

SD/MMC メモリは SD/MMC スロットに挿入して、次のコマンド：

```
# mount /dev/mmc/blk0/part1 /mnt
```

```
# ls /mnt
```

最大 2GB までの SD/MMC メモリが使えます。

2.2.4 USB カメラで撮影した画像を取る

まず、USB カメラ(中芯微又は UVC のチップを採用したものを USB ホスト又は USB ハブに挿入して、次のコマンド :

```
# ls /dev/v4l/video0(USB カメラをマウントしたのを確認します。)  
/dev/v4l/video0(この情報が見れば、USB カメラのマウントが成功しました。)  
# spccat p 100ms N 5(USB カメラの画像を取る。100ms 間 1 つ画像を取って、全ての  
5 枚 jpg ファイルを今のフォルダに保存します。)
```

2.2.5 ボタンのテスト

```
# buttons  
ARM9 ボードのユーザボタンを押してください。
```

2.2.6 シリアルポート 1/2 のテスト

TTL—RS232 変換ボード(別売)でシリアルポート 1/2 とパソコンを繋ぎます。新ハイパーターミナルを実行して、通信速度 115200、フロー制御なしを設定して、ARM9 のコンソールで次のコマンドを入力します。

```
# armcomtest -d /dev/tts/1 -o (シリアルポート 1 のテスト)  
# armcomtest -d /dev/tts/2 -o (シリアルポート 2 のテスト)
```

2.2.7 ブザーのテスト

```
# pwm_test
```

2.2.8 液晶のバックライトの制御

QQ2440v3 はこの機能がありません

```
# bl 0  
close LCD backlight  
# bl 1  
open LCD backlight
```

2.2.9 I2C デバイス(24C04)のテスト

I2C メモリ 24C04 に 0~0xFF を書き込みます。

```
# i2c -w
```

24C04 を読み出します。

```
# i2c -r
```

2.2.10 RTC の設定

(1)#date -s 042916352007 #今の時間を設定します : 2007-04-29 16:34

(2)#hwclock -w #今の時間を S3C2440 の RTC に保存します。

(3)#hwclock -s #起動の時、Linux 時間を S3C2440 の RTC から回復します。

hwclock -s コマンドは起動スクリプト(/etc/init.d/rcS)に書き込みました。起動の時、自動的に実行します。

2.2.11 液晶(LCD)画面を取ります

```
#snapshot pic.png
```

液晶(LCD)で表示された画面を pic.png ファイルとして保存します。

2.3 QQ2440v3 のネットワーク機能

2.3.1 QQ2440v3 のウェブサーバー

QQ2440v3 の Linux でウェブサーバー(boa)をインストールしました。パソコンのブラウザで <http://192.168.1.230> を入力すると、QQ2440v3 のホームページが見えます。このホームページを通じて、QQ2440v3 のユーザーLED と USB カメラをアクセスできます。

2.3.2 QQ2440v3 の Telnet と Ftp 機能

QQ2440v3 の Linux でクライアント側とサーバー側の Telnet/Ftp をインストールしました。ご利用してください。

デフォルトの設定 :

Telnet のユーザーネームは root です、password がありません。

Ftp のユーザーネームは plg です、password も plg です。

2.3.3 DNS と gateway の設定

DNS の IP アドレスを/etc/resolv.conf ファイルに書き込みます。

gateway の設定 : #route add default gw 192.168.1.1

2.3.4 MAC アドレスの設定

```
# ifconfig eth0 down
```

```
# ifconfig eth0 hw ether 00:11:AA:BB:CC:DD(新 MAC アドレス)
```

```
# ifconfig eth0 up
```

新 MAC アドレスが有効のため、これらのコマンドを起動スクリプト/etc/init.d/rcS に書き

込みます。

2.3.5 ネットワーク・ファイルシステム(NFS)のマウント

まず、ネットワーク・ファイルシステムのサーバーを構築します。(3.2 を参照してください)

```
#mount -t nfs -o nolock 192.168.1.111:/root_nfs /mnt
```

192.168.1.111 はネットワーク・ファイルシステムのサーバーの IP アドレスです。

マウント成功すれば、ARM9 は大きなリモート・ハードディスク(/mnt)を直接にアクセスできます。プログラムを開発する時が便利です。

```
#umount /mnt #リモート・ハードディスクを ARM9 システムから外します。
```

第三章 Linux のクロス開発環境

3.1 クロス開発環境を構築

QQ2440v3 のクロス開発環境は Redhat9.0 をお勧めします。ほかの Linux の場合は、エラーが出すかもしれません。

Redhat9.0 で ARM のクロス開発ツールをインストールします。

```
# tar xvzf arm-linux-gcc-3.3.2.tgz -C /
```

```
# tar xvzf arm-linux-gcc-2.95.3.tgz -C /
```

```
# tar xvzf arm-linux-gcc-3.4.1.tgz -C /
```

使いやすいため、「export PATH=\$PATH:/usr/local/arm/3.4.1/bin」を.bashrc ファイルに入れます。

再び Redhat9 にログイン、次のコマンド：

```
# arm-linux-gcc -v
```



```
root@localhost:~  
File Edit View Terminal Go Help  
root@localhost:~ root@localhost:~  
[root@localhost root]# arm-linux-gcc -v  
Reading specs from /usr/local/arm/3.4.1/lib/gcc/arm-linux/3.4.1/specs  
Configured with: /work/crosstool-0.27/build/arm-linux/gcc-3.4.1-glibc-2.3.2/gcc-3.4.1/configure --target=arm-linux --host=i686-host_pc-linux-gnu --prefix=/usr/local/arm/3.4.1 --with-headers=/usr/local/arm/3.4.1/arm-linux/include --with-local-prefix=/usr/local/arm/3.4.1/arm-linux --disable-nls --enable-threads=posix --enable-symvers=gnu --enable-__cxa_atexit --enable-languages=c,c++ --enable-shared --enable-c99 --enable-long-long  
Thread model: posix  
gcc version 3.4.1  
[root@localhost root]#
```

この画面が出たら、ARM 用のクロス開発環境をインストール成功しました。他のクロス開発ツールを使用したら、例えば 2.95.3、「export PATH=\$PATH:/usr/local/arm/2.95.3/bin」を.bashrc ファイルに入れます。

3.2 NFS サーバーを構築

ネットワーク・ファイルシステム(NFS)を使用すれば、ARM9 は大きなりモート・ハードディスクを直接にアクセスできます。プログラムを開発する時が便利です。次は NFS サーバーを構築手順です。

(1) NFS のルートシステムファイルを解凍します。

```
# tar xvzf root_nfs.tgz -C /
```

(2) /etc/exports ファイルを編集します。

「/root_nfs *(rw, sync, no_root_squash)」を/etc/exports に入れます。

(3) NFS サーバを起動します。

```
#/etc/init.d/nfs start
```

(4) NFS ファイルシステムを確認します。

```
#mount -t nfs localhost:/root_nfs /mnt/
```

```
#ls /mnt
```

firewall の設定によって、NFS へアクセスできない可能性があります。

3.3 NFS はルートファイルシステムとして起動

ARM9 ボードが起動、又はリセットの時、ハイパーターミナルにスペースキーを押します。次のコマンドを入力します。

```
Supervivi>param set linux_cmd_line "console=ttySAC0 root=/dev/nfs  
nfsroot=192.168.1.111:/root_nfs
```

```
ip=192.168.1.70:192.168.1.111:192.168.1.111:255.255.255.0:QQ2440v3.arm9.net:eth0:of  
f"
```

param set linux_cmd_line は linux 起動のパラメーターです。パラメーターの意味は：

nfsroot は NFS サーバーの IP アドレス。

“ ip= ” の後ろ：

第一項(192.168.1.70)は ARM9 ボードの IP ；

第二項(192.168.1.111)はホストの IP ；

第三項(192.168.1.111)はゲットウェイの IP ；

第四項(255.255.255.0)はネットマスク ；

第五項はホストのドメイン(自由的に入力でも大丈夫です)

eth0 は LAN デバイスの名前。

Boot コマンドを入力すると

```
Supervivi>boot
```

ARM9 ボードは NFS からブートします。

第四章 Linux のアプリケーションを開発

4.1 Hello, World!

Hello, World のソースコードは examples.tgz にあります。

```
#include <stdio.h>
int main(void) {
    printf("hello, FriendlyARM!\n");
}
```

4.2 Hello,World をコンパイル

```
#cd examples/hello
```

```
#arm-linux-gcc -o hello main.c
```

又は

```
#make
```

実行できるhelloを生成します。

4.2 Hello,World を ARM9 ボードで実行

生成された実行コードhelloをARM9ボードに入れて、ARM9のコンソールで実行します。

```
# ./hello
```

```
hello, FriendlyARM!
```

ARM9ボードに入れるのは幾つの方法があります。USBメモリ、SDメモリ、FTPなど。一番便利な方法はNFSです。ARM9ボードは直接にホスト側の実行ファイルを実行できます。

4.3 他のサンプル

examples.tgzに幾つのサンプルがあります。



examples/led: LEDテスト

examples/buttons: ボタンのテスト

examples/udpak: UDPサンプル

examples/math: 数学処理

examples/pthread: マルチthreadサンプル

examples/led-player: pipeサンプル

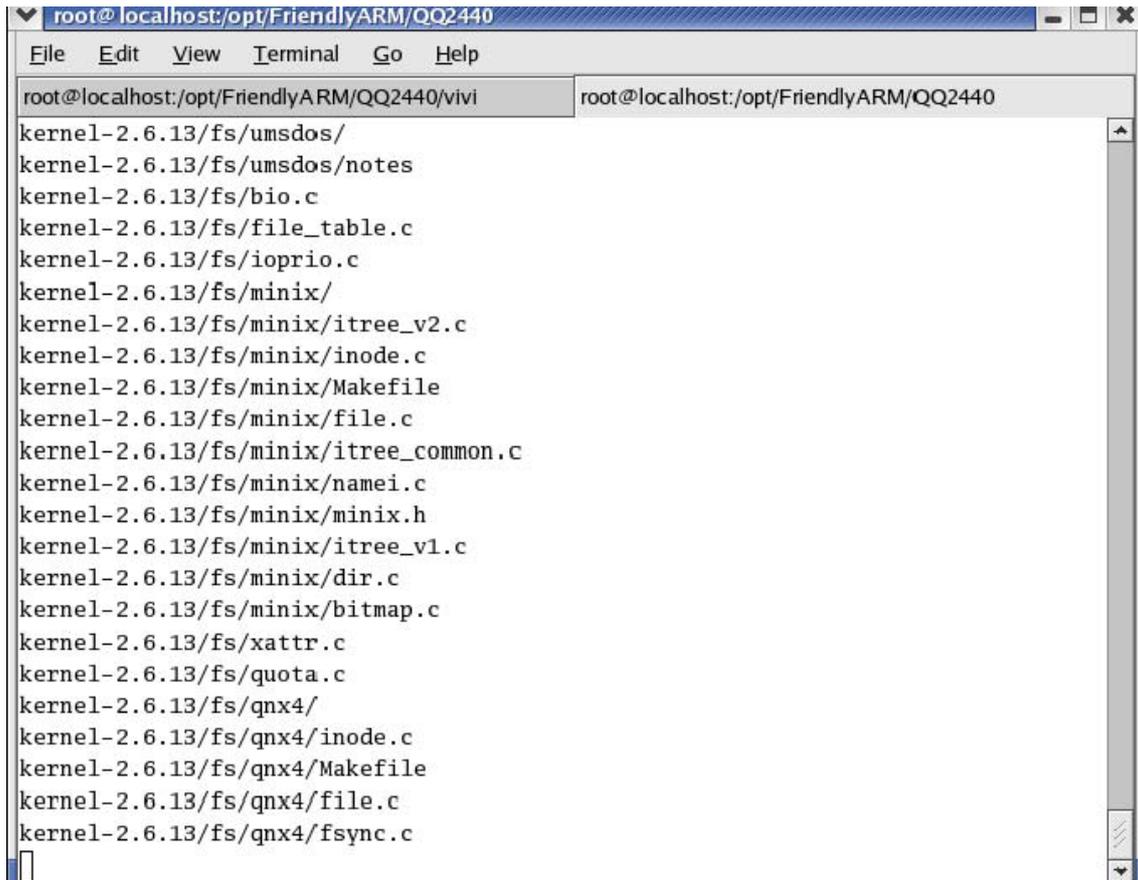
examples/c++: c++のサンプル

第五章 Linux カーネルを再構築

5.1 カーネルのソースコードを解凍

```
#tar xvzf linux-2.6.13-2440-20080910.tgz
```

このカーネルはQQ2440v3/MINI2440に両対応する

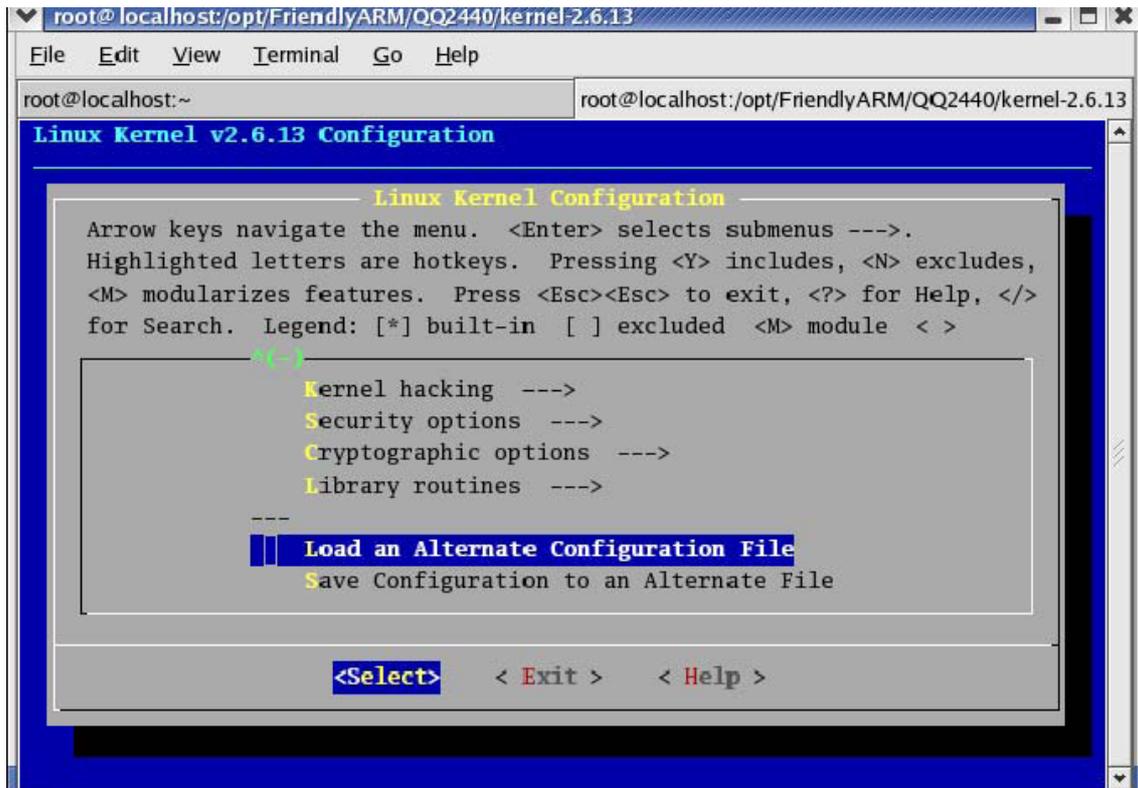


```
root@localhost:/opt/FriendlyARM/QQ2440
File Edit View Terminal Go Help
root@localhost:/opt/FriendlyARM/QQ2440/vivi root@localhost:/opt/FriendlyARM/QQ2440
kernel-2.6.13/fs/umsdos/
kernel-2.6.13/fs/umsdos/notes
kernel-2.6.13/fs/bio.c
kernel-2.6.13/fs/file_table.c
kernel-2.6.13/fs/ioprio.c
kernel-2.6.13/fs/minix/
kernel-2.6.13/fs/minix/itree_v2.c
kernel-2.6.13/fs/minix/inode.c
kernel-2.6.13/fs/minix/Makefile
kernel-2.6.13/fs/minix/file.c
kernel-2.6.13/fs/minix/itree_common.c
kernel-2.6.13/fs/minix/namei.c
kernel-2.6.13/fs/minix/minix.h
kernel-2.6.13/fs/minix/itree_v1.c
kernel-2.6.13/fs/minix/dir.c
kernel-2.6.13/fs/minix/bitmap.c
kernel-2.6.13/fs/xattr.c
kernel-2.6.13/fs/quota.c
kernel-2.6.13/fs/qnx4/
kernel-2.6.13/fs/qnx4/inode.c
kernel-2.6.13/fs/qnx4/Makefile
kernel-2.6.13/fs/qnx4/file.c
kernel-2.6.13/fs/qnx4/fsync.c
```

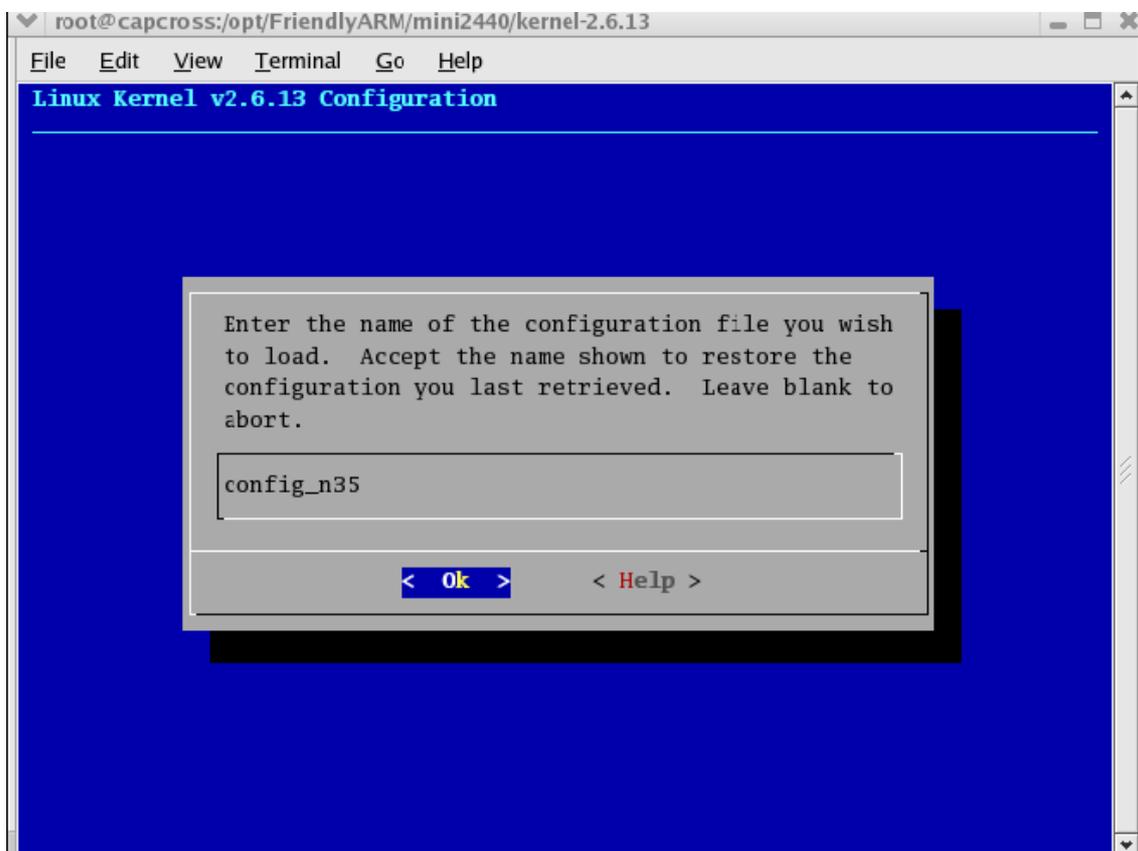
5.2 コンフィグ

```
# cd kernel-2.6.13
```

```
# make menuconfig
```



メニューの「Load an Alternate Configuration File」を選択してください。



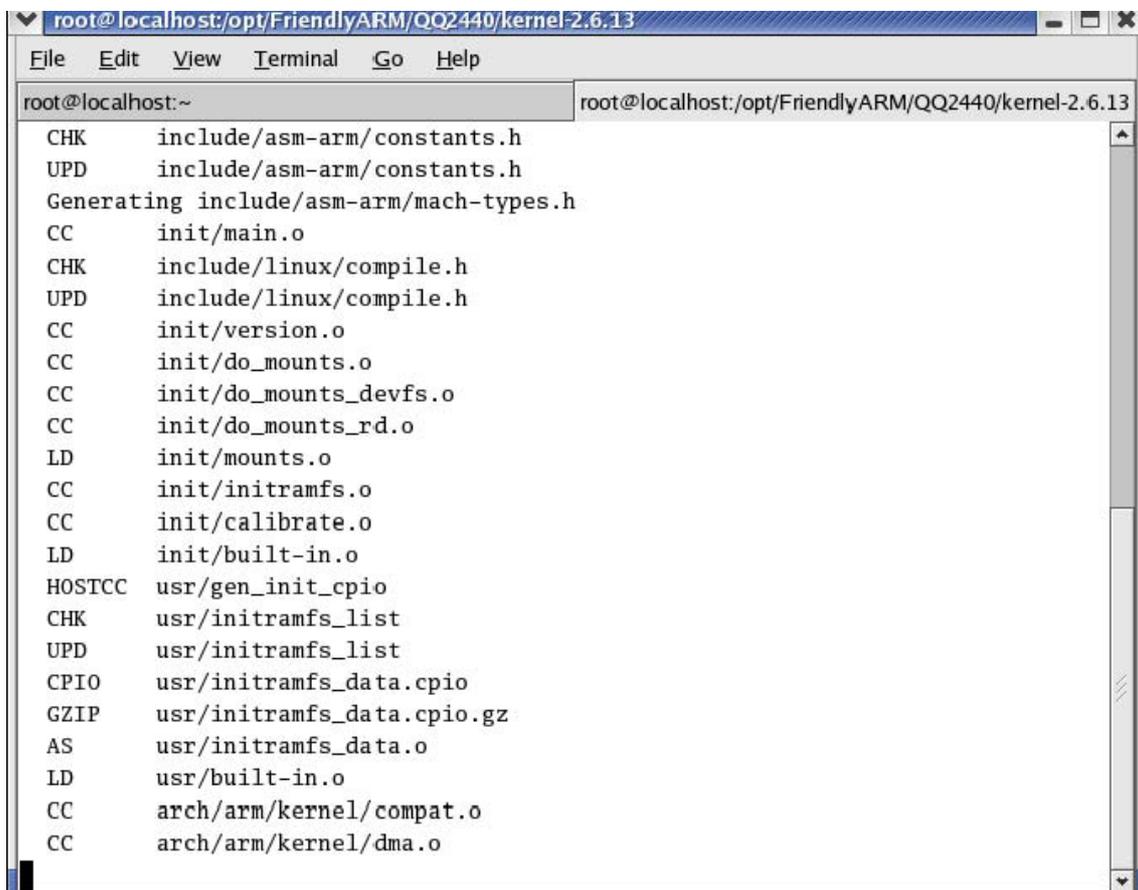
コンフィグファイル名前「config_n35」を入力して、「ok」ボタンを押します。

Kernel-2.6.13に幾つのコンフィグファイルがあります。名前は「config_xxx_yyyy」です。xxxは「n35」、又は「a70」です。「n35」は3.5”液晶に対応する。「a70」は7”液晶に対応する。yyyyは「qq2440」、又は「mini2440」です。「qq2440」はQQ2440v3ボードに対応する。「mini2440」はMINI2440ボードに対応する。

5.3 カーネルをコンパイル

カーネルをコンパイルするため、**arm-linux-gcc-3.4.1**が必要です。

```
# make zImage
```



```
root@localhost:~/opt/FriendlyARM/qq2440/kernel-2.6.13
File Edit View Terminal Go Help
root@localhost:~ root@localhost:~/opt/FriendlyARM/qq2440/kernel-2.6.13
CHK include/asm-arm/constants.h
UPD include/asm-arm/constants.h
Generating include/asm-arm/mach-types.h
CC init/main.o
CHK include/linux/compile.h
UPD include/linux/compile.h
CC init/version.o
CC init/do_mounts.o
CC init/do_mounts_devfs.o
CC init/do_mounts_rdt.o
LD init/mounts.o
CC init/initramfs.o
CC init/calibrate.o
LD init/built-in.o
HOSTCC usr/gen_init_cpio
CHK usr/initramfs_list
UPD usr/initramfs_list
CPIO usr/initramfs_data.cpio
GZIP usr/initramfs_data.cpio.gz
AS usr/initramfs_data.o
LD usr/built-in.o
CC arch/arm/kernel/compat.o
CC arch/arm/kernel/dma.o
```

コンパイル完了したら、arch/arm/bootしたでカーネルファイルzImageを生成させます。

5.4 ドライバの場所

(1)DM9000 Ethernet devices

kernel-2.6.13/drivers/net/dm9000x.c

(2)シリアルポート dev/tts/0,1,2

kernel-2.6.13/drivers/serial/s3c2410.c

(3) RTC

kernel-2.6.13/drivers/char/s3c2410-rtc.c

(4)LED

kernel-2.6.13/drivers/char/mini2440_leds.c

(5)ボタン

kernel-2.6.13/drivers/char/mini2440_buttons.c

(6)タッチパネル

kernel-2.6.13/drivers/input/touchscreen/s3c2410_ts.c

(7)yaffs ファイルシステム

kernel-2.6.13/fs/yaffs2

(8)USBマウス、キーボード

kernel-2.6.13/drivers/usb/input/hid-input.c

(9)SD/MMCドライバ、linux-2.6.13は2GBまでのメモリをアクセスできます。

kernel-2.6.13/drivers/mmc

(10)Nand Flash

kernel-2.6.13/drivers/mtd/nand

(11)UDA1341オーディオ

kernel-2.6.13/sound/oss/uda1341.c

kernel-2.6.13/drivers/l3

(12)LCD

kernel-2.6.13/drivers/video/s3c2410fb.c

(13)USBメモリ

kernel-2.6.13/drivers/usb/storage

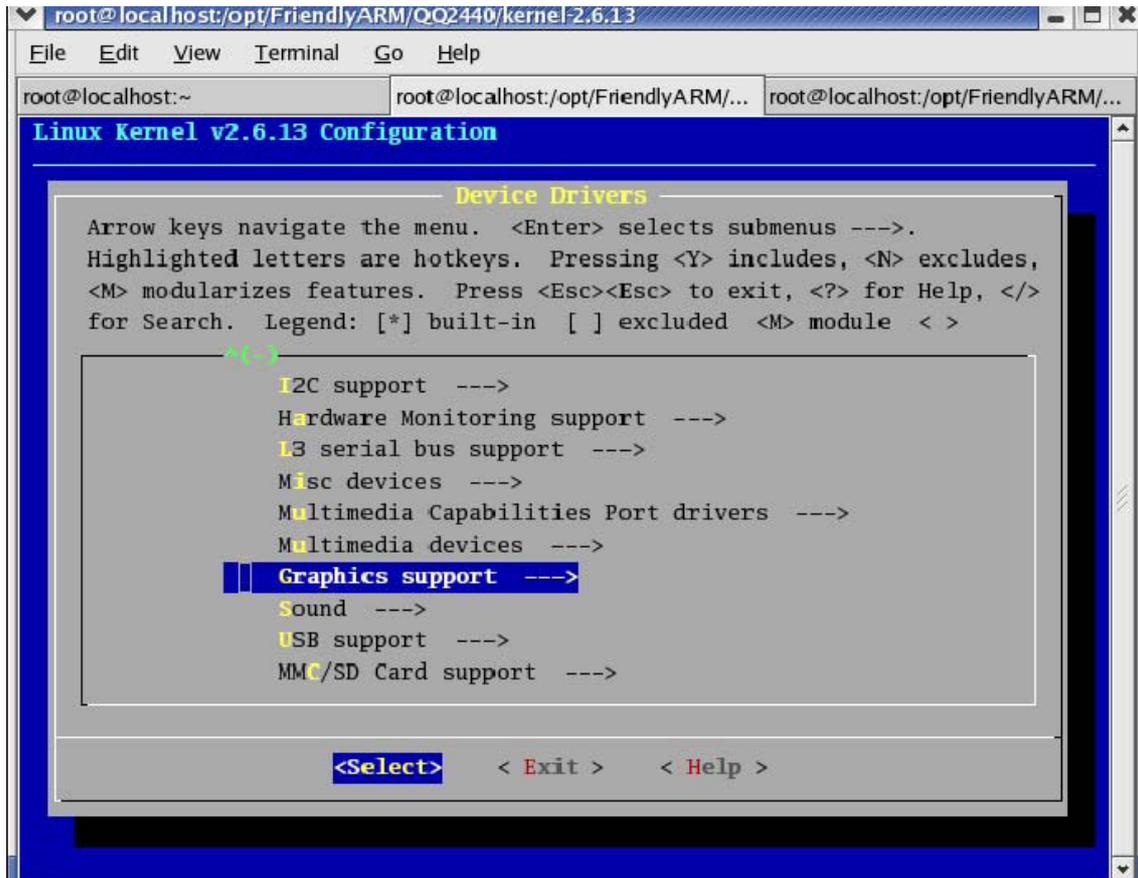
(14)USBカメラ

kernel-2.6.13/drivers/usb/media/gspca

5.5 デバイスドライバのコンフィグ

5.5.1 LCD のコンフィグ

メインメニューで「Device Drivers」を選択します。

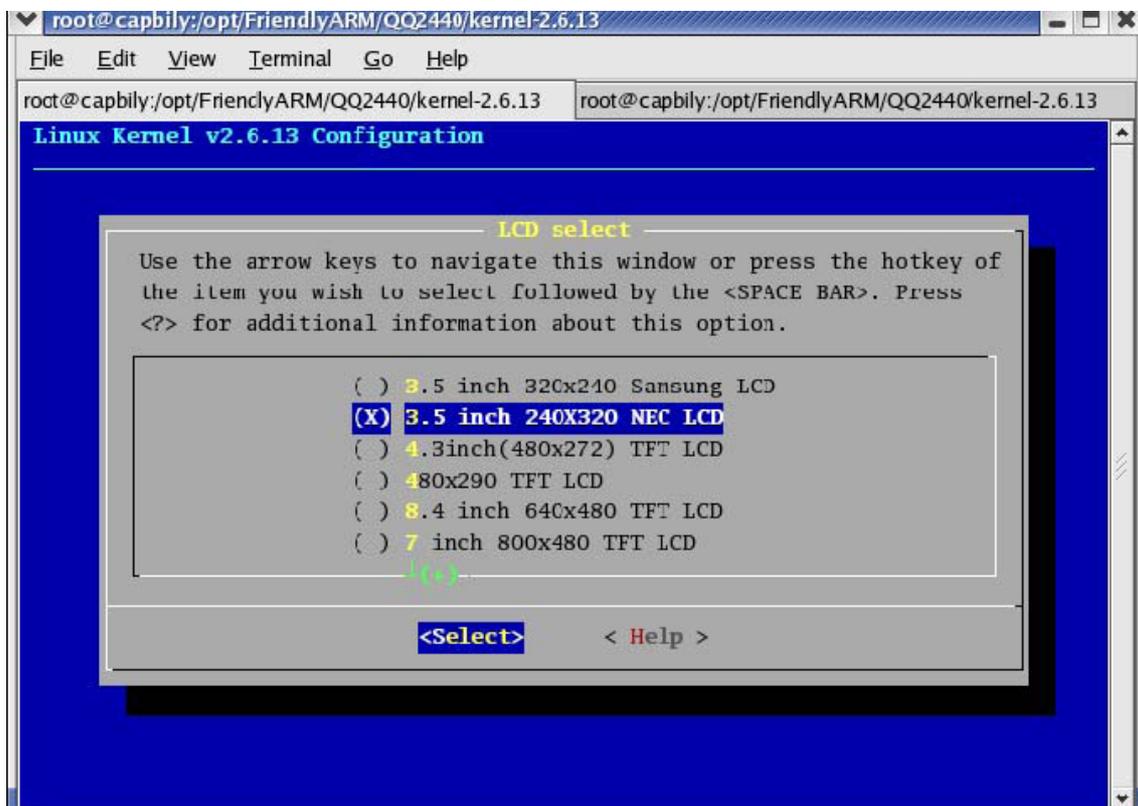
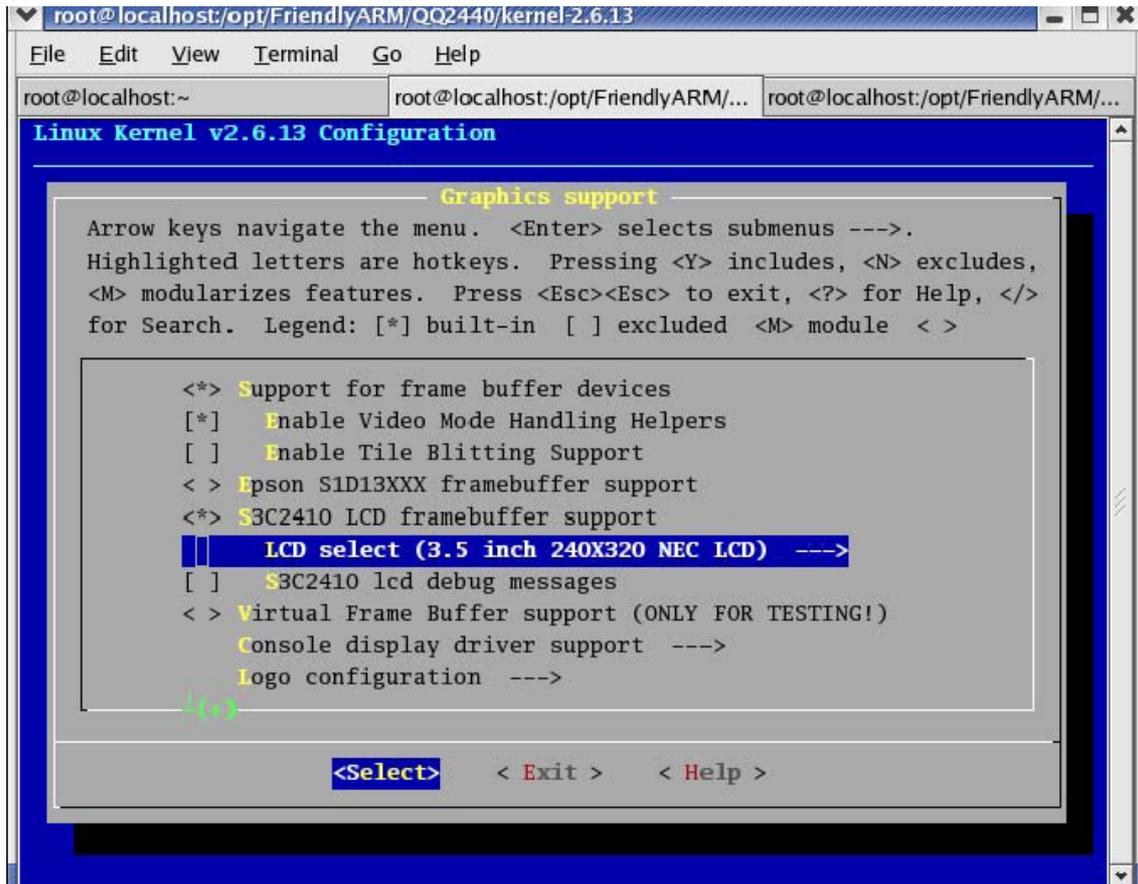


「Graphics support」を選択して、「*」を入れます。

<*> Support for frame buffer devices

<*> S3C2410 LCD framebuffer support

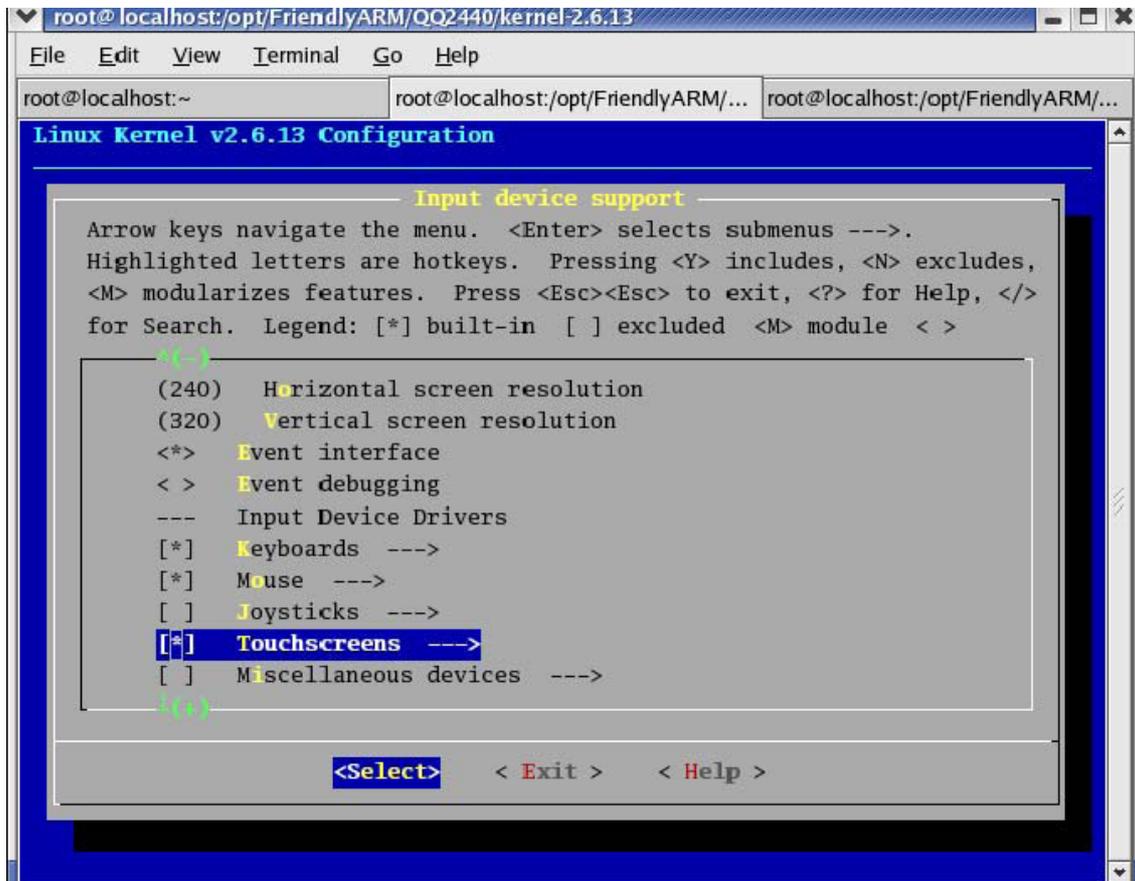
「LCD select」を選択します。



対応されたLCDドライバを選択してください。

5.5.2 タッチパネルのコンフィグ

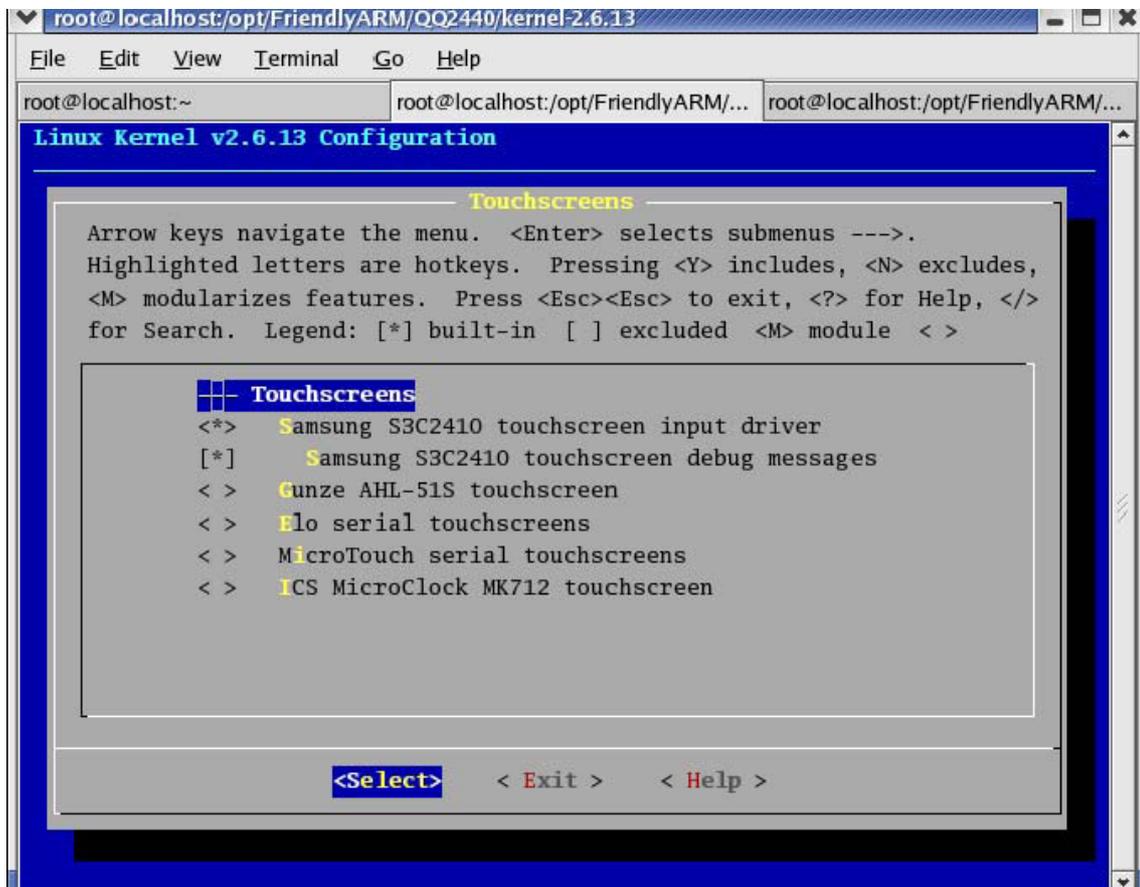
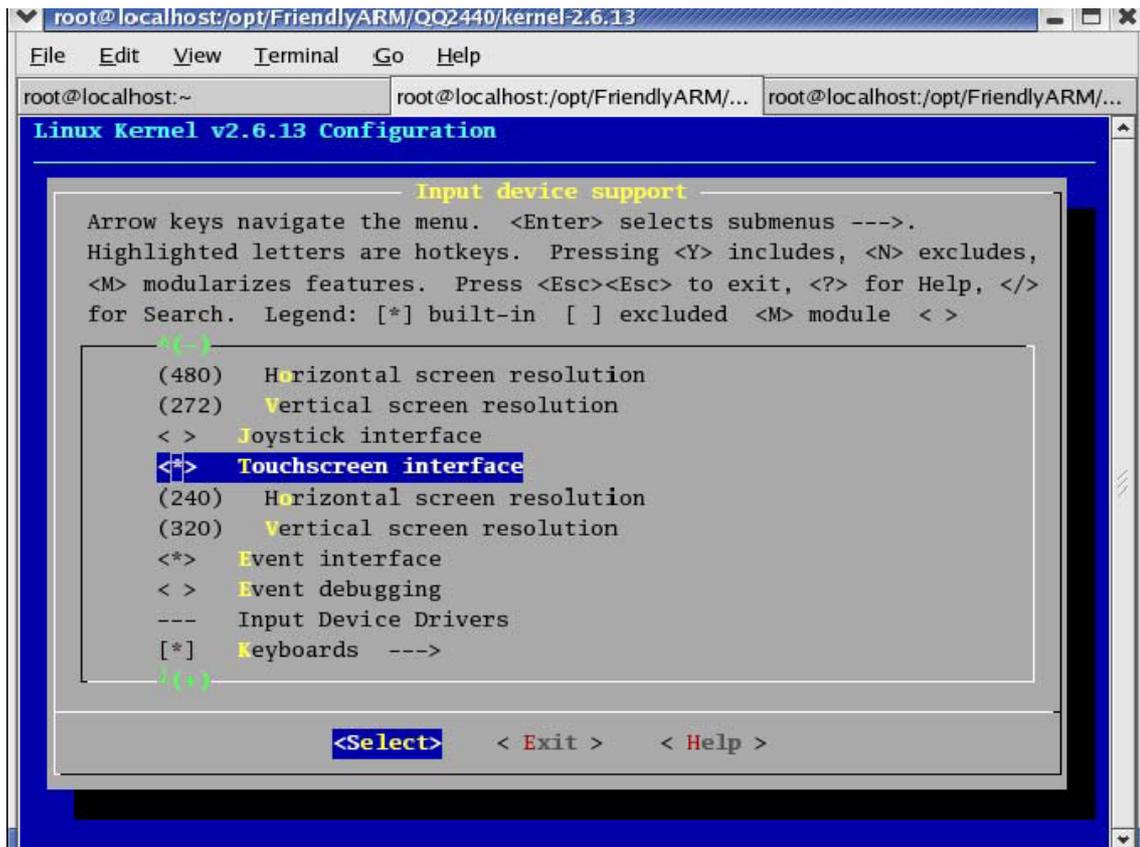
「Device Drivers」メニューで「input device support」を選択します。



「Touchscreens」に入ります。次を選択します。

<*> Touchscreen interface

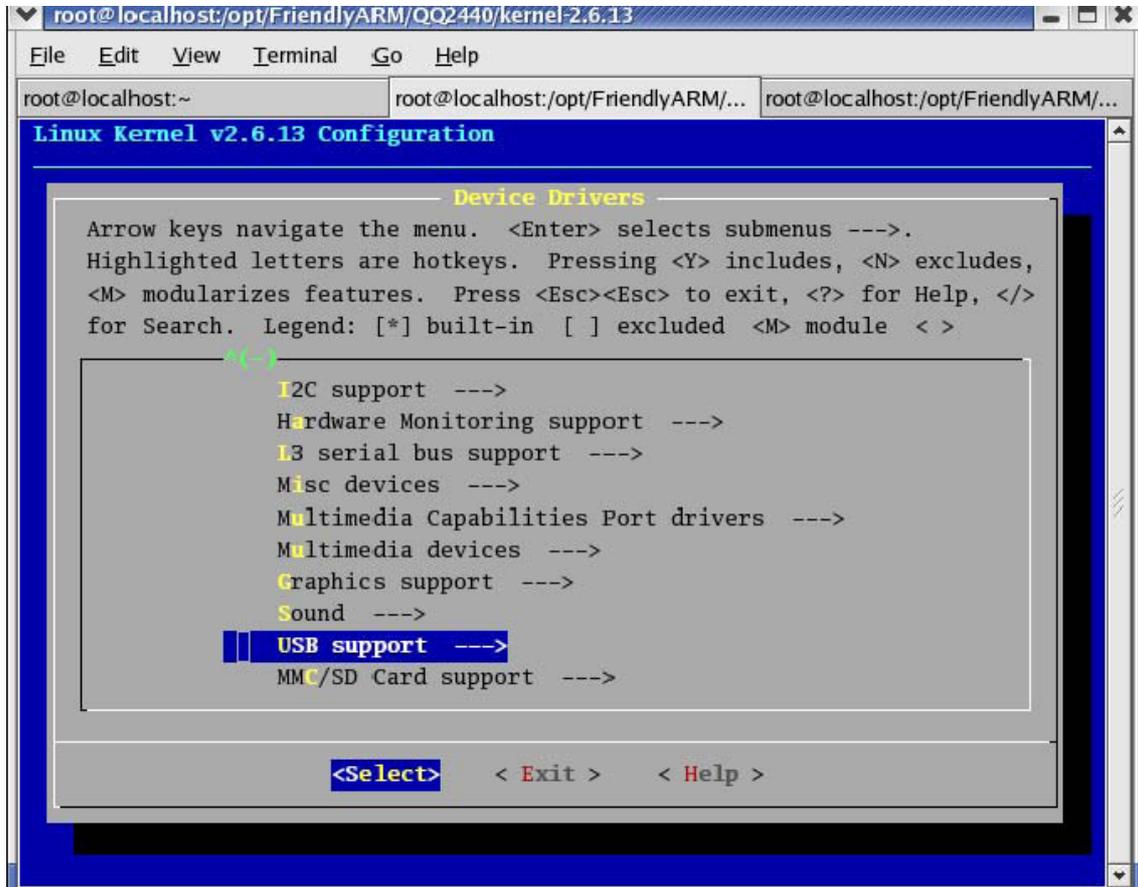
<*> Event interface



5.5.3 USB マウスとキーボード

「Device Drivers」を選択して、次に「*」を入れます。

<*>USB support



次に「*」を入れます。

<*> Support for Host-side USB

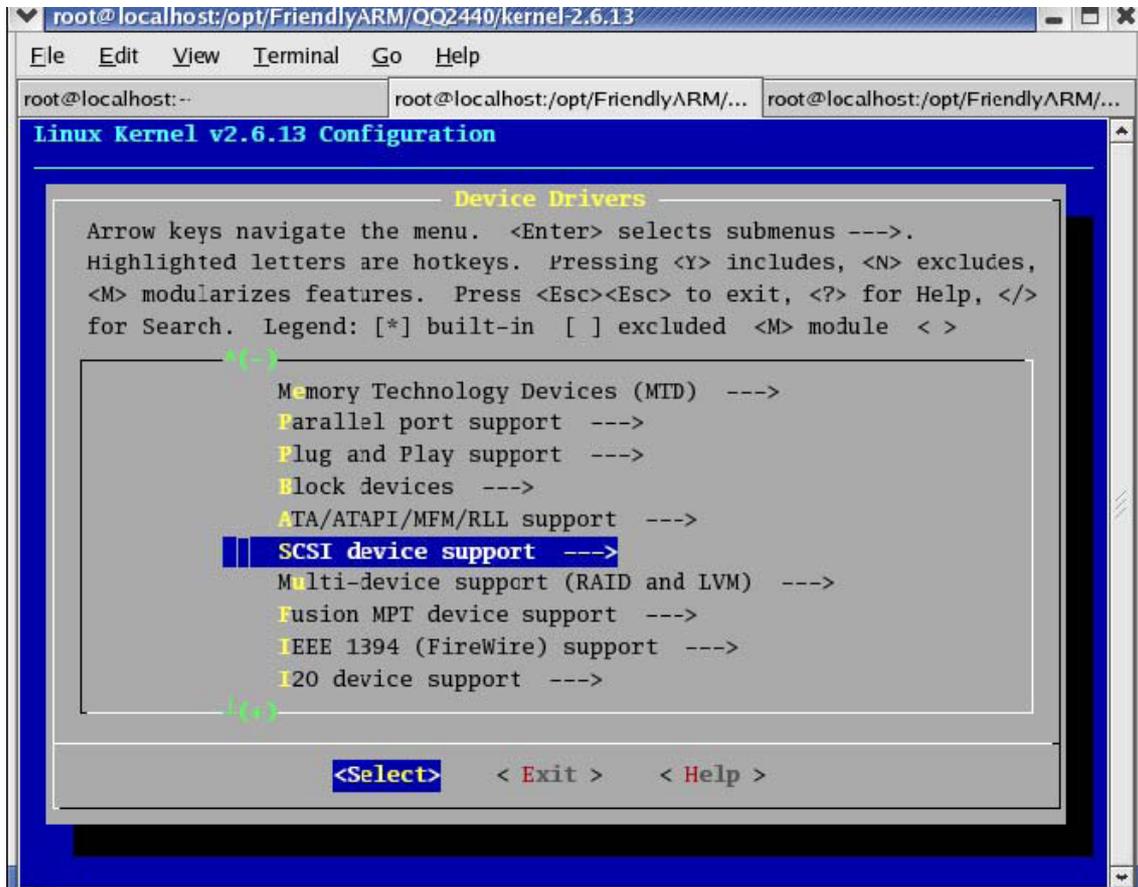
<*> OHCI HCD support

```
root@localhost:/opt/FriendlyARM/qq2440/kernel-2.6.13
File Edit View Terminal Go Help
root@localhost:~ root@localhost:/opt/FriendlyARM/... root@localhost:/opt/FriendlyARM/...
Linux Kernel v2.6.13 Configuration
----- USB support -----
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >
<?> Support for Host-side USB
[ ] USB verbose debug messages
--- Miscellaneous USB options
[ ] USB device filesystem
[ ] Enforce USB bandwidth allocation (EXPERIMENTAL)
[ ] Dynamic USB minor allocation (EXPERIMENTAL)
[ ] USB suspend/resume (EXPERIMENTAL)
--- USB Host Controller Drivers
< > ISP116X HCD support
<*> OHCI HCD support
[?>]
<Select> < Exit > < Help >
```

```
root@localhost:/opt/FriendlyARM/qq2440/kernel-2.6.13
File Edit View Terminal Go Help
root@localhost:~ root@localhost:/opt/FriendlyARM/... root@localhost:/opt/FriendlyARM/...
Linux Kernel v2.6.13 Configuration
----- USB support -----
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >
[?>]
[ ] SanDisk SDDR-55 SmartMedia support (EXPERIMENTAL)
[ ] Lexar Jumpshot Compact Flash Reader (EXPERIMENTAL)
[?>] USB Input Devices
<*> USB Human Interface Device (full HID) support
[*] HID input layer support
[ ] Force feedback support (EXPERIMENTAL)
[ ] /dev/hiddev raw HID device support
< > Aiptek 6000U/8000U tablet support
< > Wacom Intuos/Graphire tablet support
< > Acecad Flair tablet support
[?>]
<Select> < Exit > < Help >
```

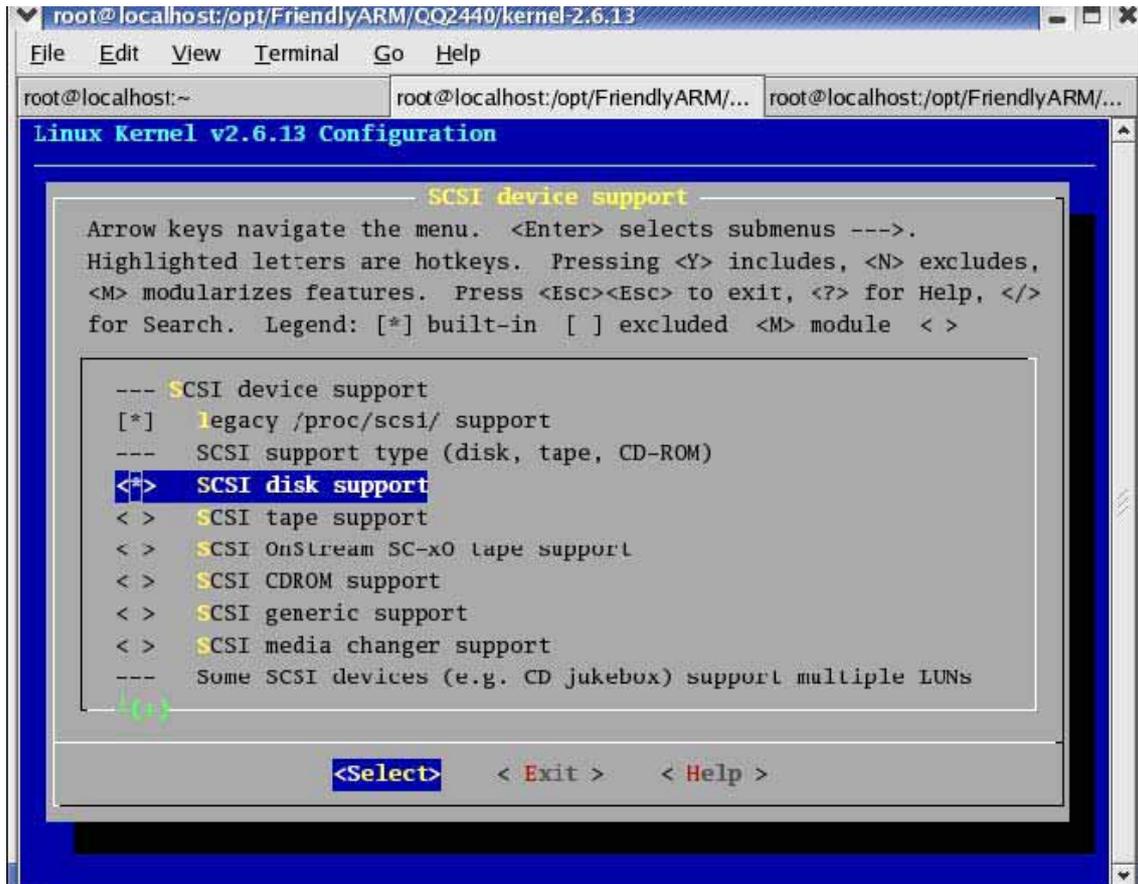
5.5.4 USB メモリ

「Device Drivers」 → 「SCSI device support」



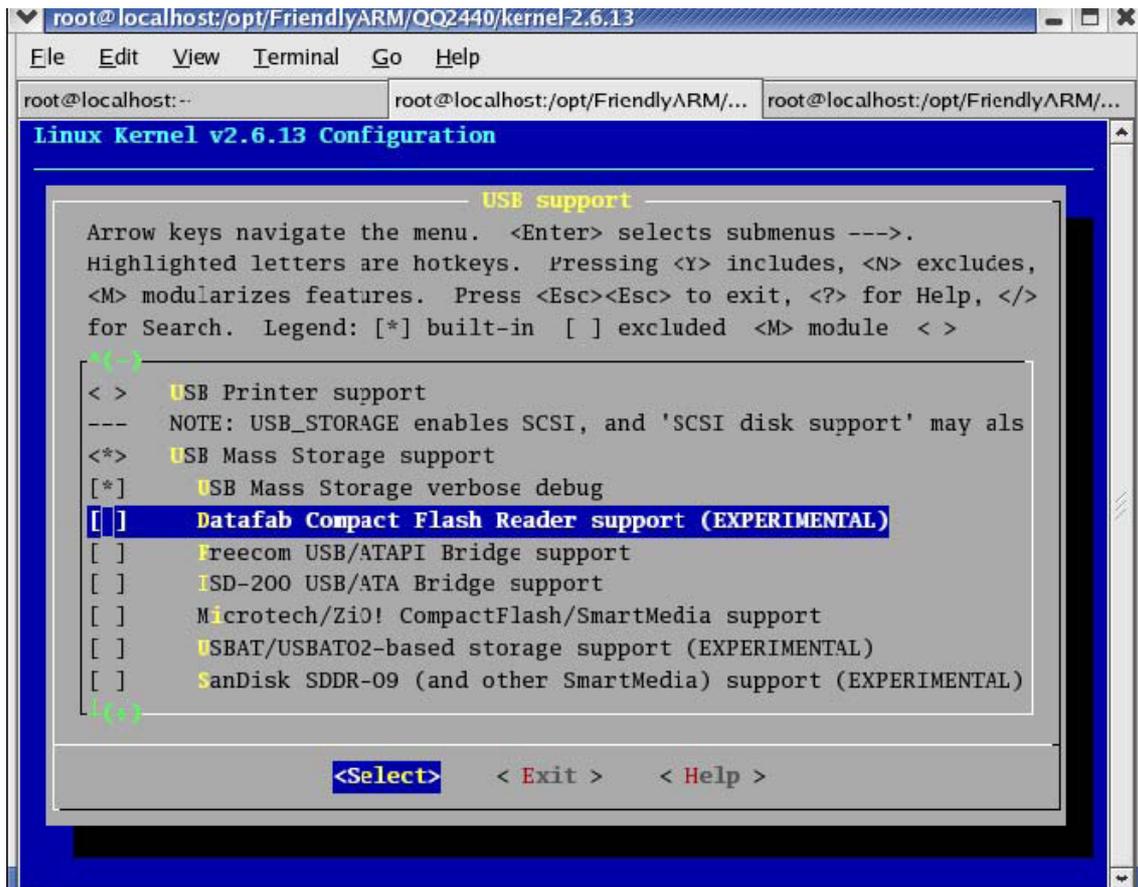
[*] legacy /proc/scsi support

<*> SCSI disk support



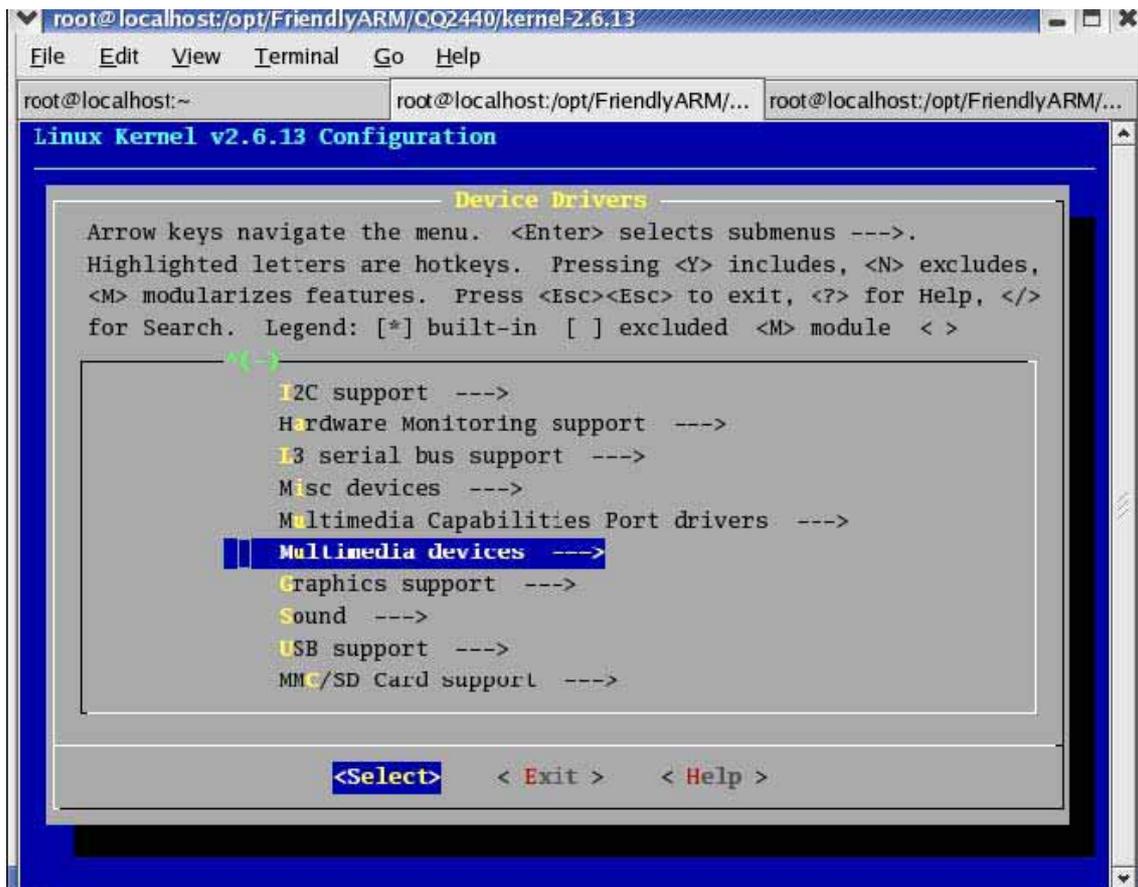
メニュー「Device Drivers」→「USB support」

<*> USB Mass Storage support

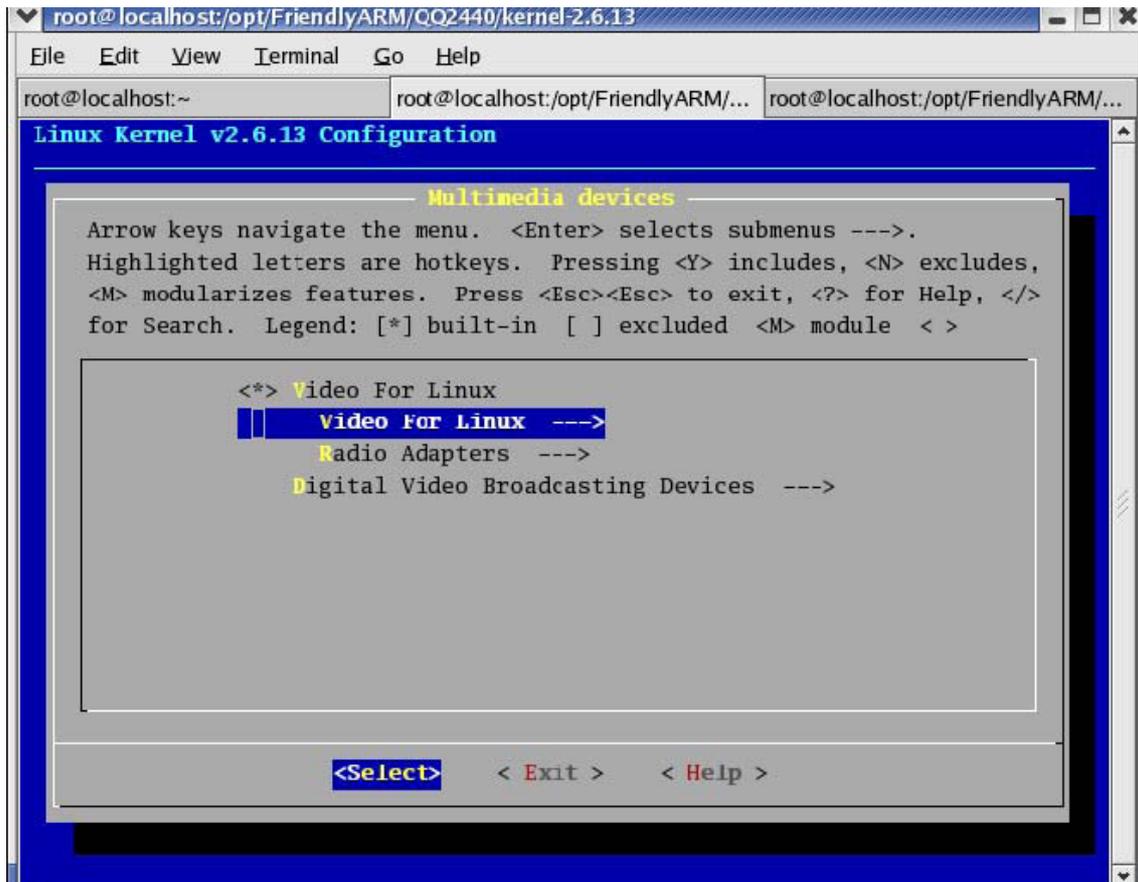


5.5.5 USB カメラ

メニュー「Device Drivers」→「Multimedia devices」



<*> Video For Linux

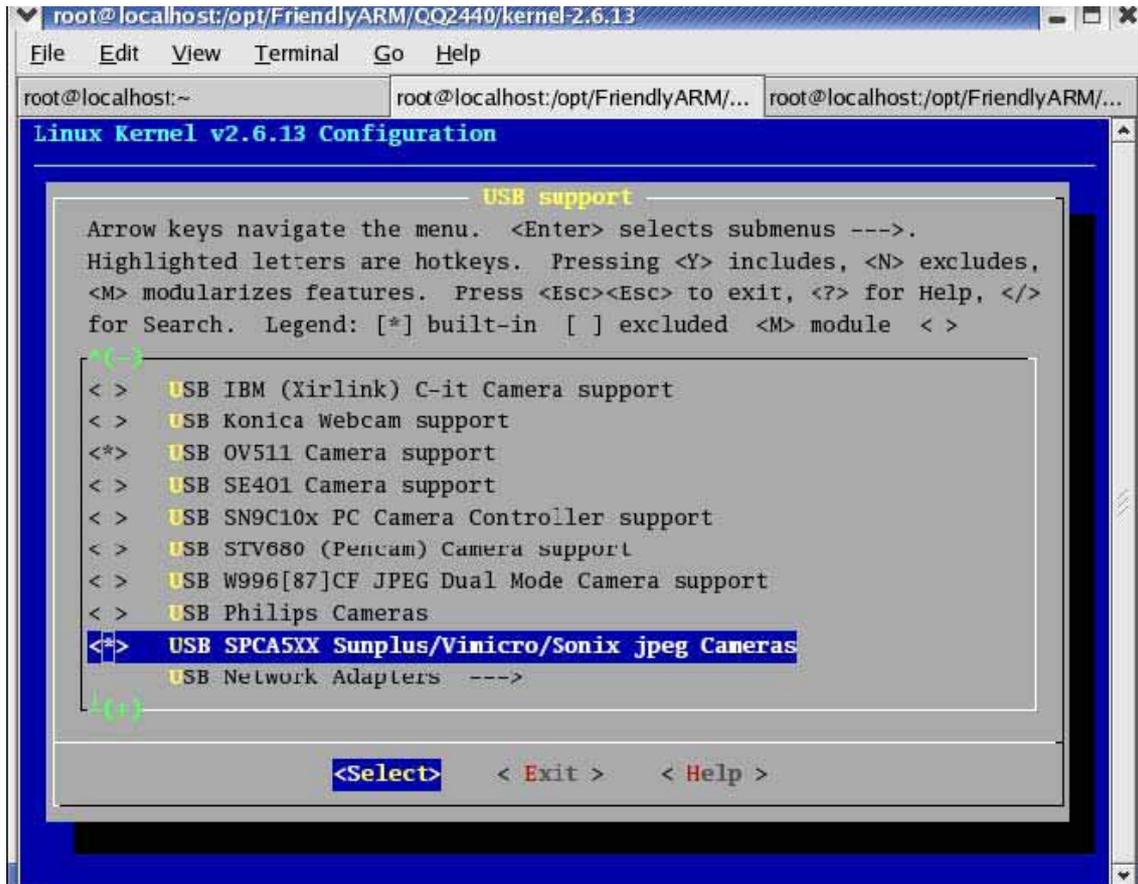


<*> OmniVision Camera Chip support

メニュー - 「Device Drivers」 → 「USB support」

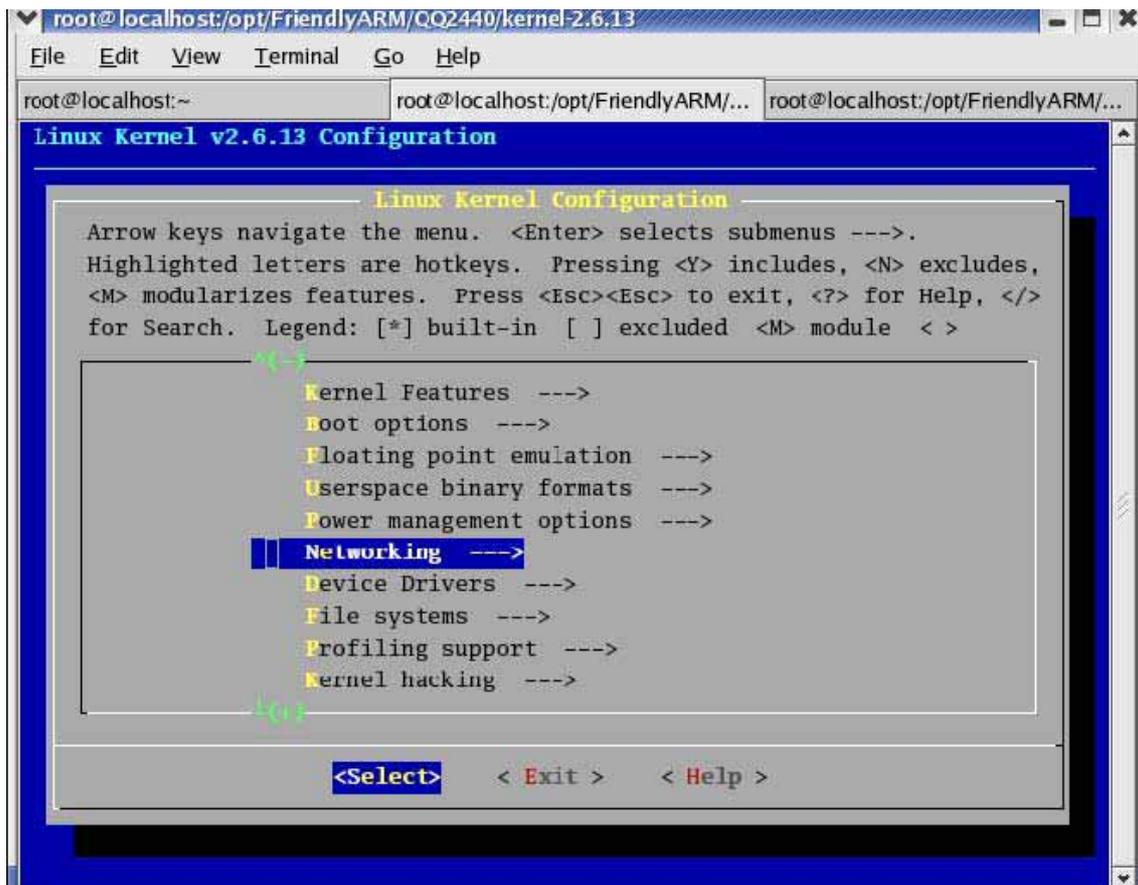
<*> USB OV511 Camera support

<*> USB SPCA5XX Sunplus/Vimicro/Sonix jpeg Cameras



5.5.6 CS8900 Ethernet

メインメニューで「Networking」を選択します。



[*] Networking support

「Networking options」に入ります。

```
root@localhost:/opt/FriendlyARM/QQ2440/kernel-2.6.13
File Edit View Terminal Go Help
root@localhost:~ root@localhost:/opt/FriendlyARM/... root@localhost:/opt/FriendlyARM/...
Linux Kernel v2.6.13 Configuration

Networking options
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <>

<*> Packet socket
[*] Packet socket: mmaped IO
<*> Unix domain sockets
<> PF_KEY sockets
[*] TCP/IP networking
[ ] IP: multicasting
[ ] IP: advanced router
[*] IP: kernel level autoconfiguration
[ ] IP: DHCP support
[*] IP: BOOTP support

<Select> <Exit> <Help>
```

```
root@localhost:/opt/FriendlyARM/QQ2440/kernel-2.6.13
File Edit View Terminal Go Help
root@localhost:~ root@localhost:/opt/FriendlyARM/... root@localhost:/opt/FriendlyARM/...
Linux Kernel v2.6.13 Configuration

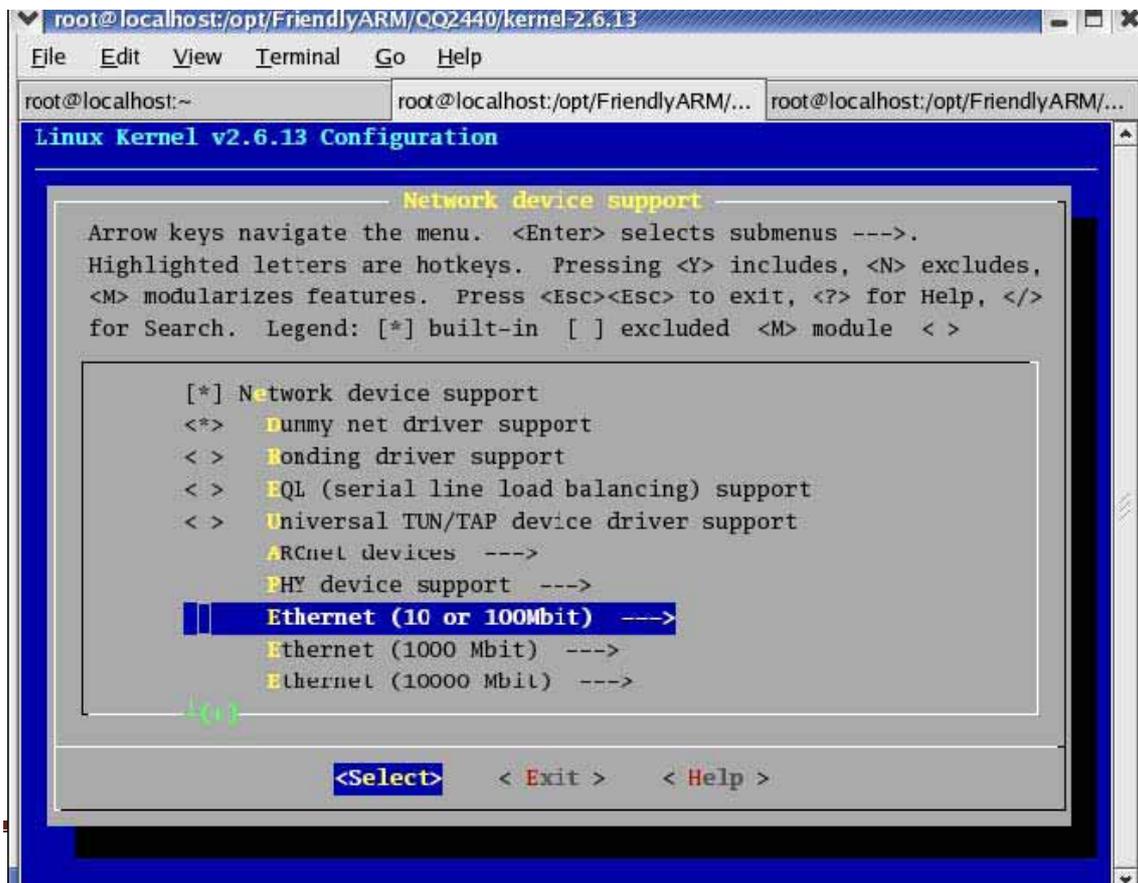
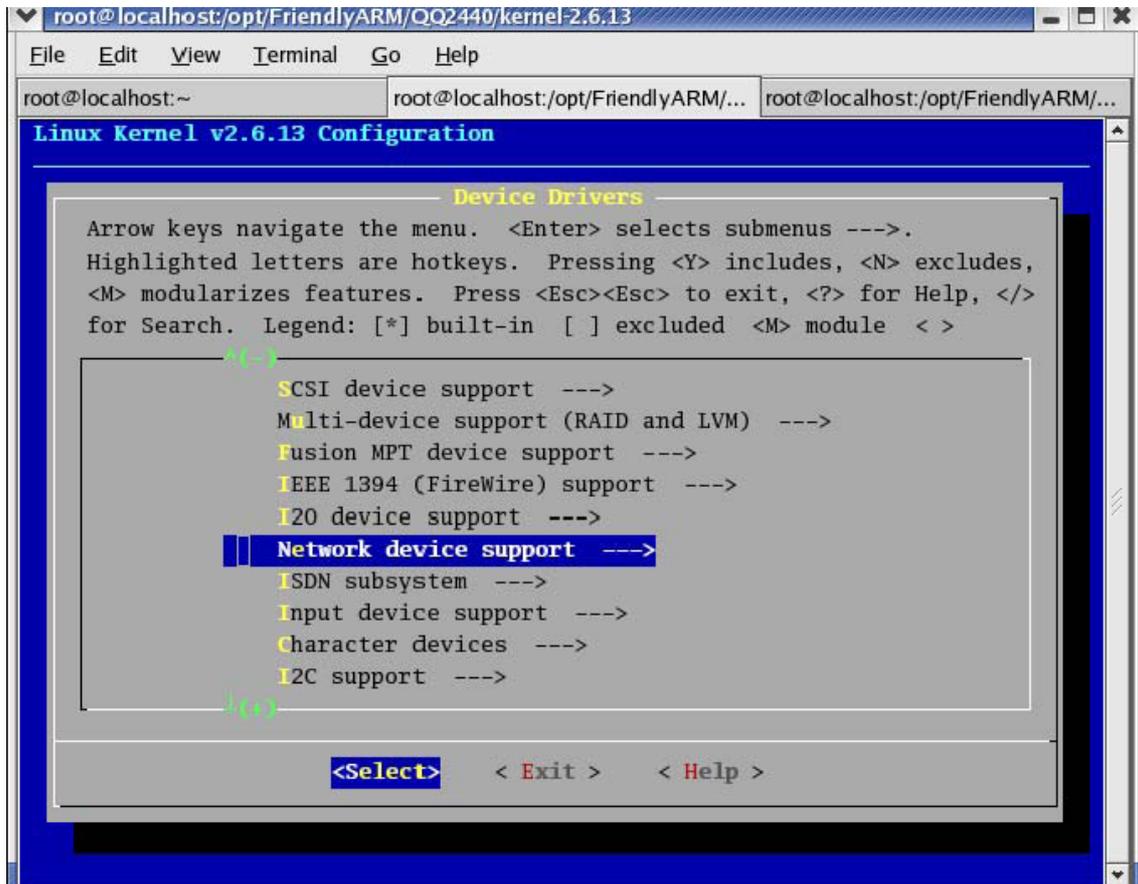
Networking
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <>

[*] Networking support
[ ] Networking options --->
[ ] Amateur Radio support --->
<> IrDA (infrared) subsystem support --->
<> Bluetooth subsystem support --->

<Select> <Exit> <Help>
```

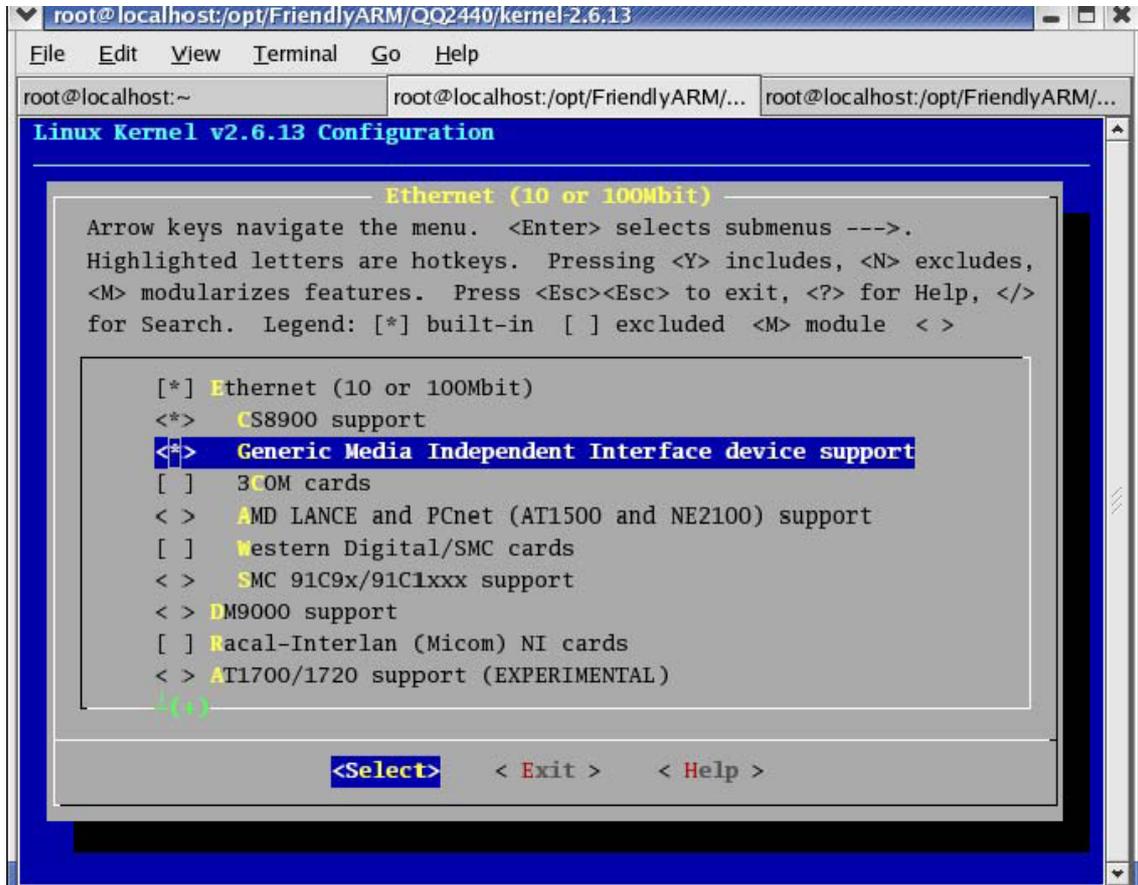


メニュー「Device Drivers」→「Network device support」



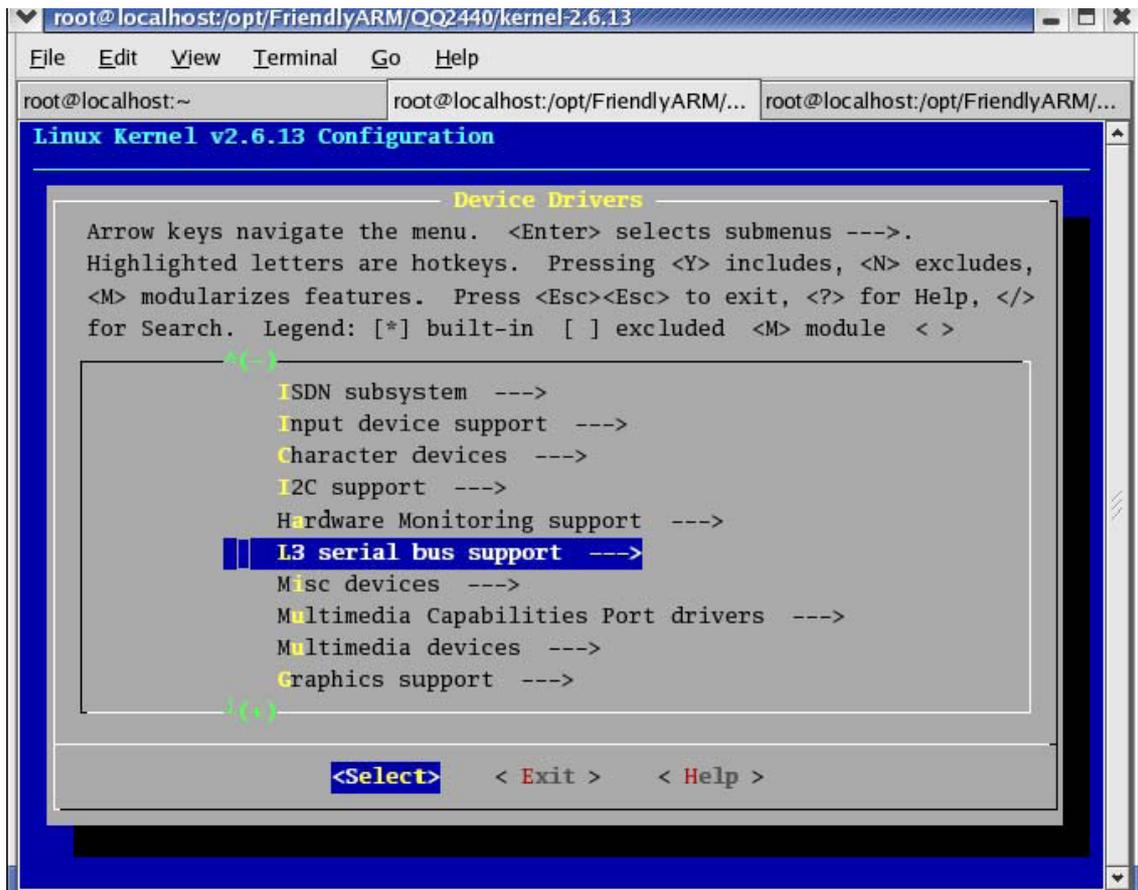
<*> CS8900 support

<*> Generic Media Independent Interface device support



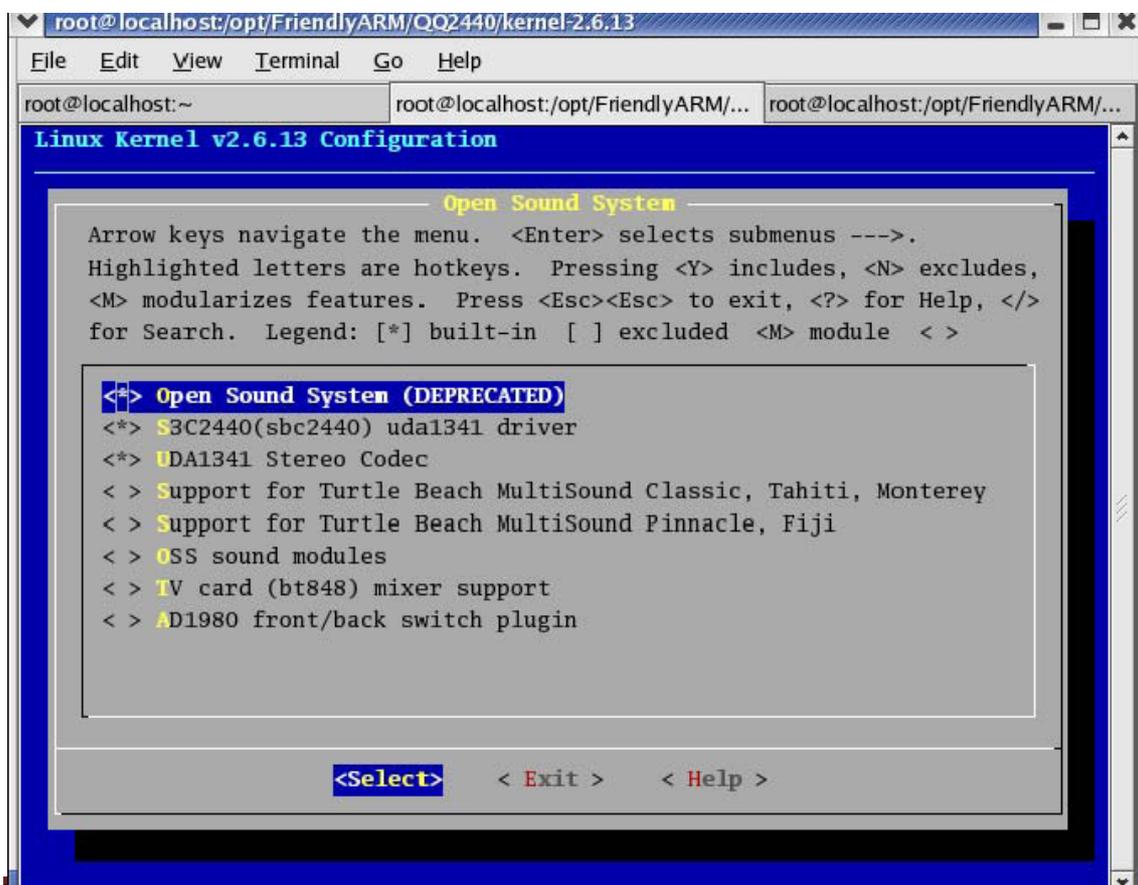
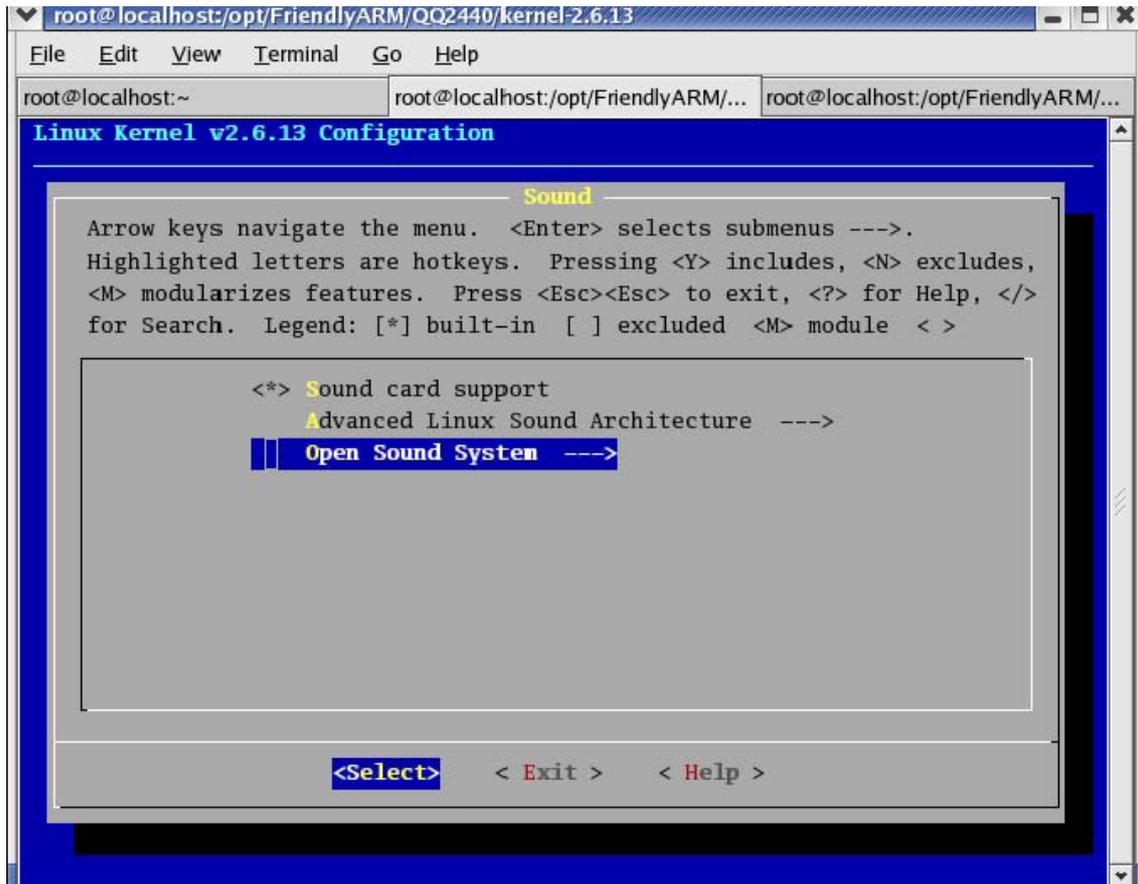
5.5.7 オーディオ

メニュー「Device Drivers」→「L3 serial bus support」



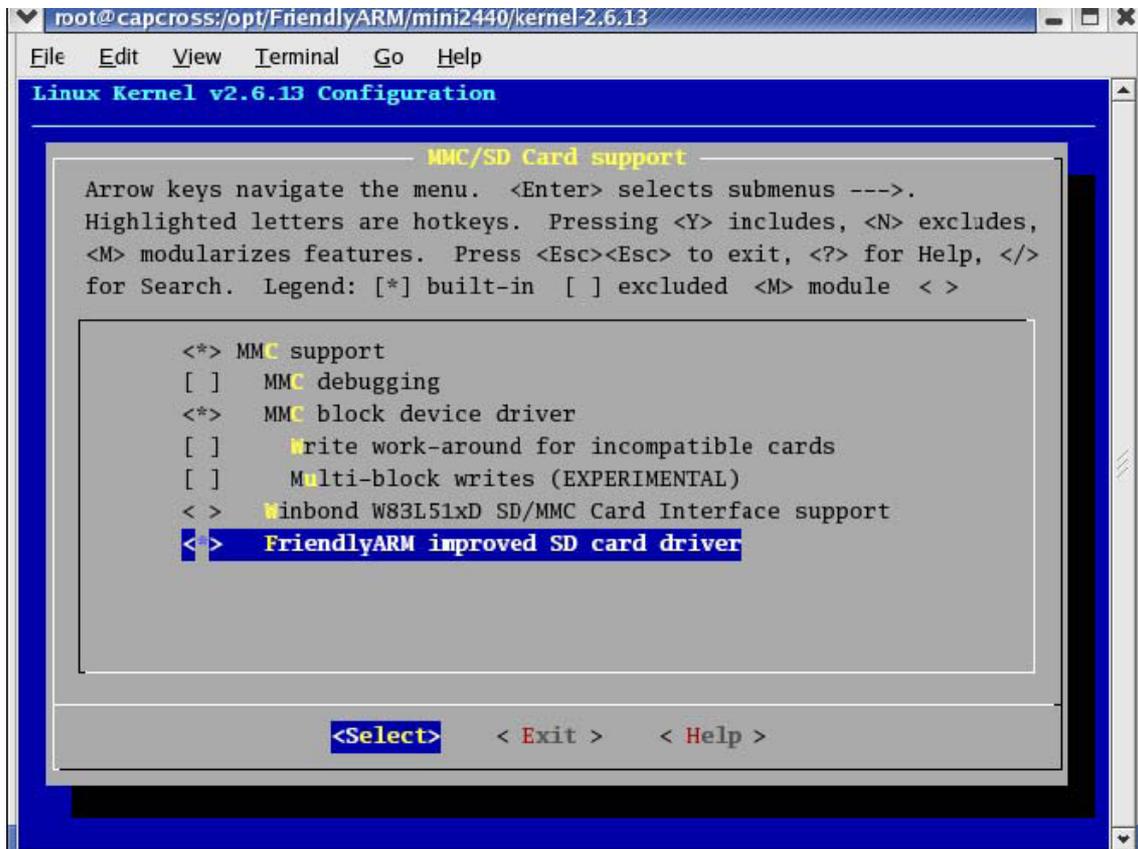
「L3 serial bus support」中の内容を全部選択します。

メニュー「Device Drivers」→「Sound→」
Open Sound System - ->に入ります。



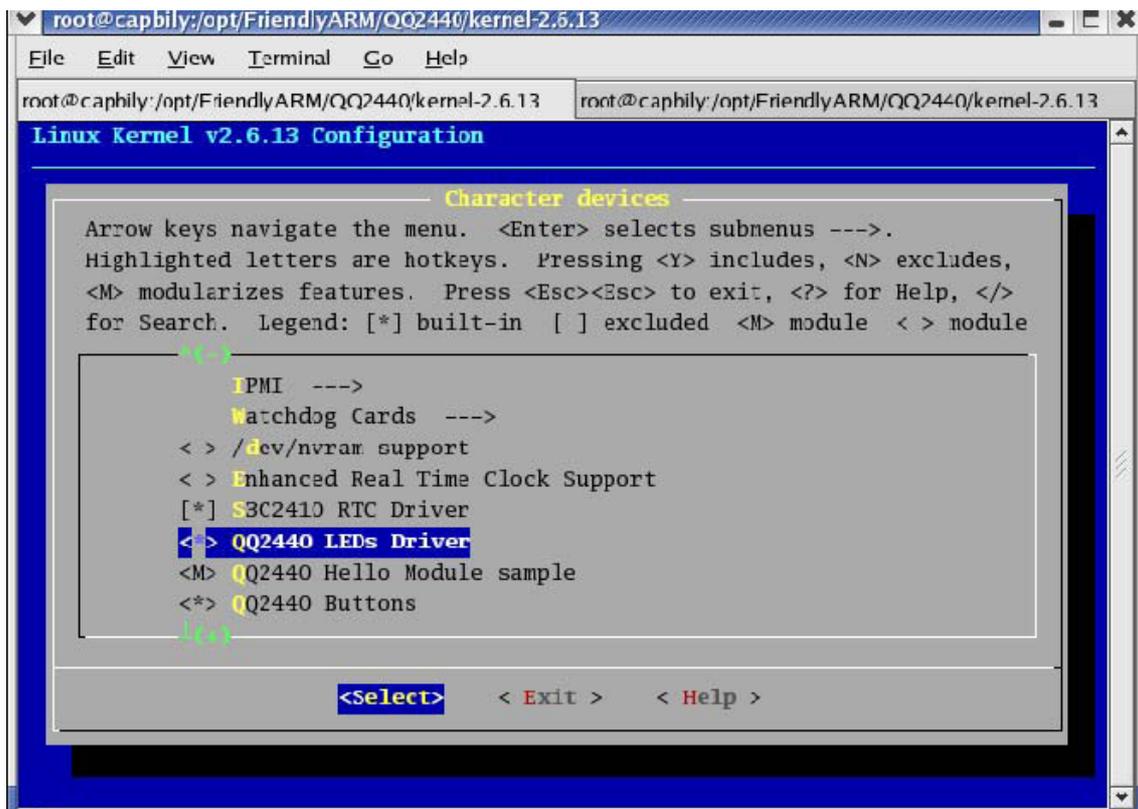
5.5.8 SD/MMC

メニュー「Device Drivers」→「MMC/SD Card support - - ->」



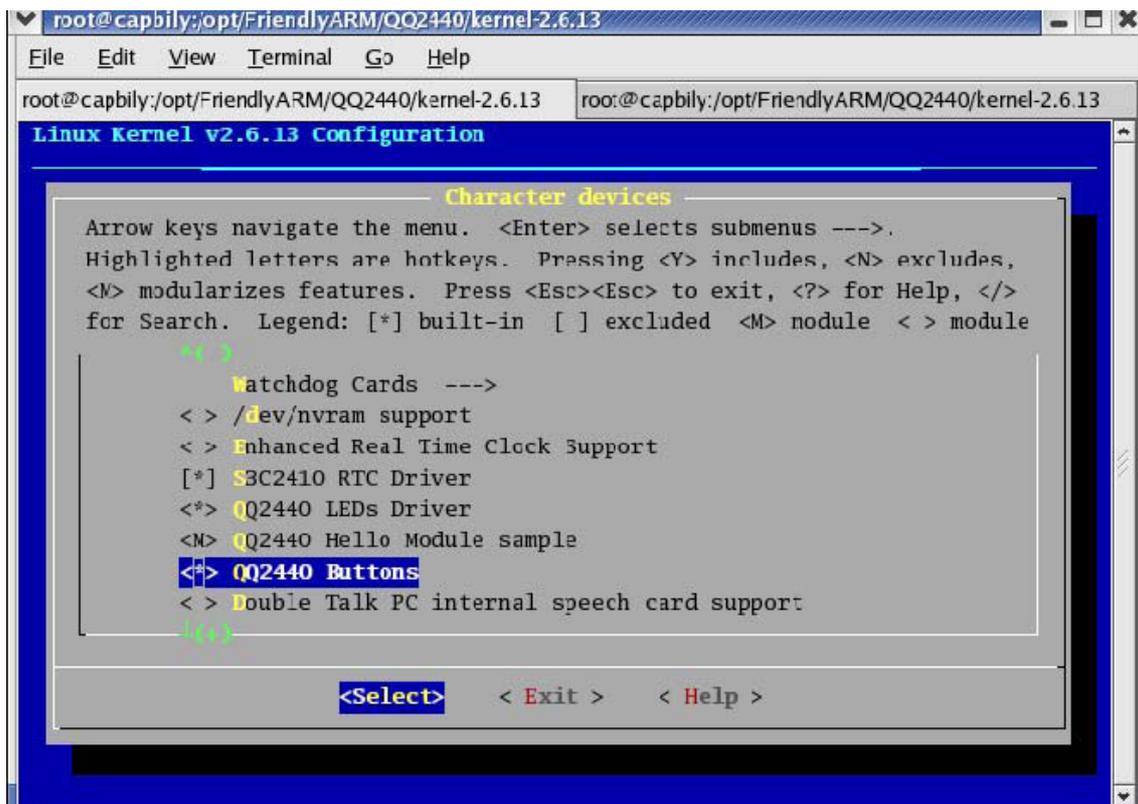
5.5.9 LED

メニュー「Device Drivers」→「Character devices - - ->」



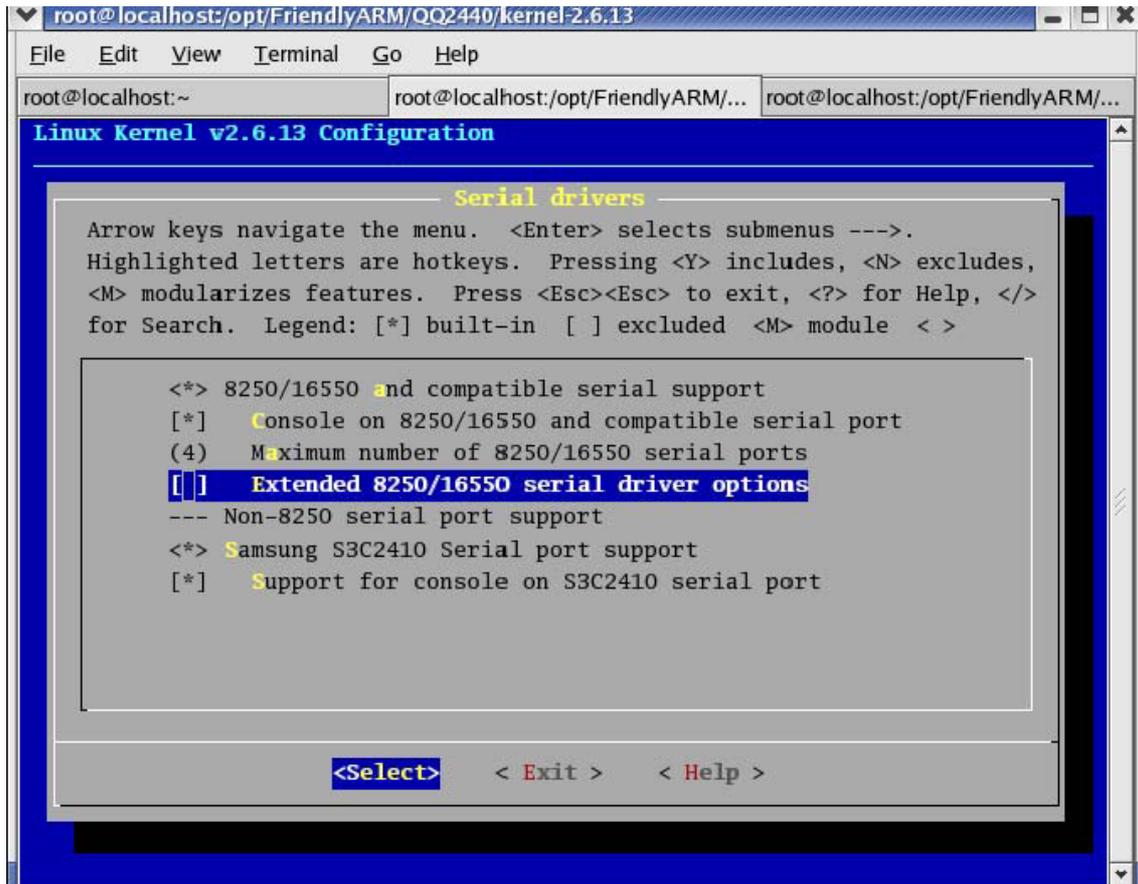
5.5.10 ボタン

メニュー「Device Drivers」→「Character devices --->」



5.5.11 シリアルポート

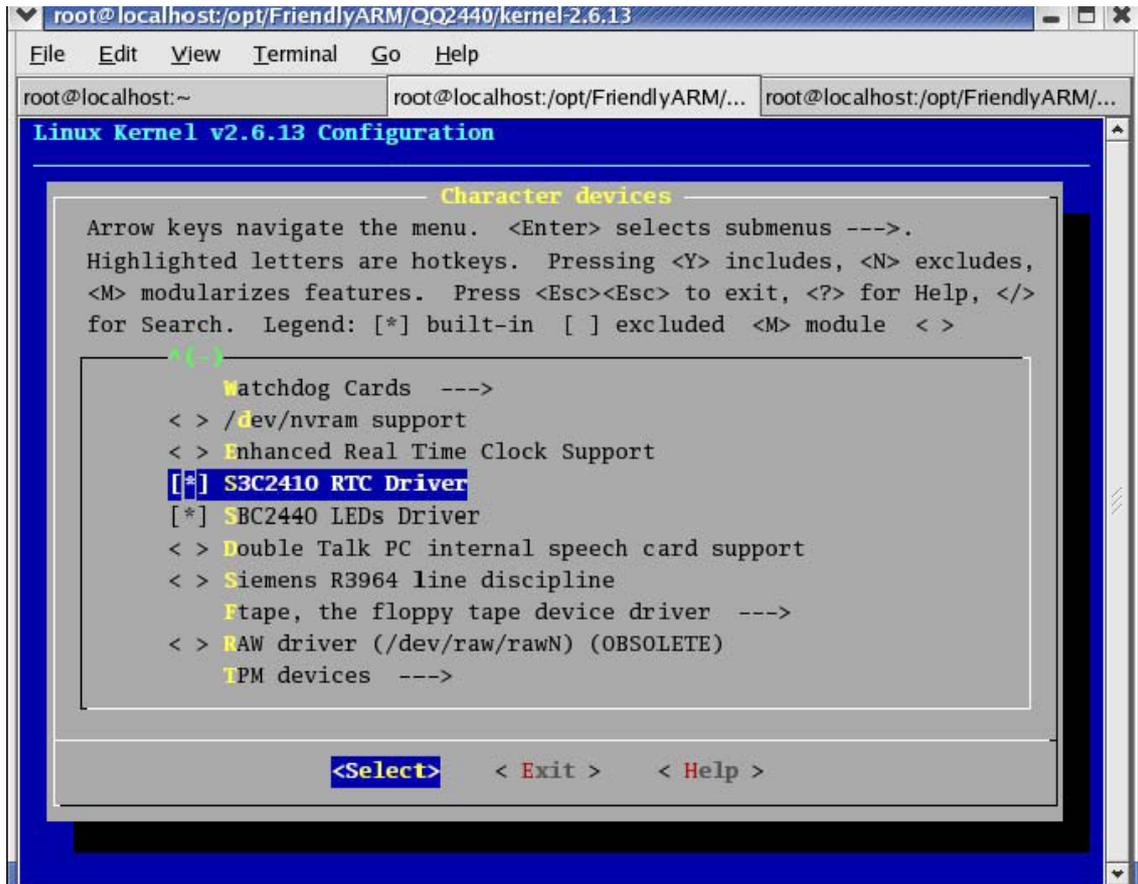
メニュー「Device Drivers」→「Character devices --->」→「Serial drivers --->」



5.5.12 RTC

メニュー「Device Drivers」→「Character devices --->」

「*Enhanced Real Time Clock Support*」を選択しません



5.5.13 yaffs ファイルシステム

メニュー「File Systems -->」→「Miscellaneous filesystems --->」

```
root@localhost:/opt/FriendlyARM/qq2440/kernel-2.6.13
File Edit View Terminal Go Help
root@localhost:~ root@localhost:/opt/FriendlyARM/... root@localhost:/opt/FriendlyARM/...
Linux Kernel v2.6.13 Configuration

File systems
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >

[*] Inotify file change notification support
[ ] Quota support
<*> Kernel automounter support
<*> Kernel automounter version 4 support (also supports v3)
CD-ROM/DVD Filesystems --->
DOS/FAT/NT Filesystems --->
Pseudo filesystems --->
Miscellaneous filesystems --->
Network File Systems --->
Partition Types --->

<Select> < Exit > < Help >
```

```
root@localhost:/opt/FriendlyARM/qq2440/kernel-2.6.13
File Edit View Terminal Go Help
root@localhost:~ root@localhost:/opt/FriendlyARM/... root@localhost:/opt/FriendlyARM/...
Linux Kernel v2.6.13 Configuration

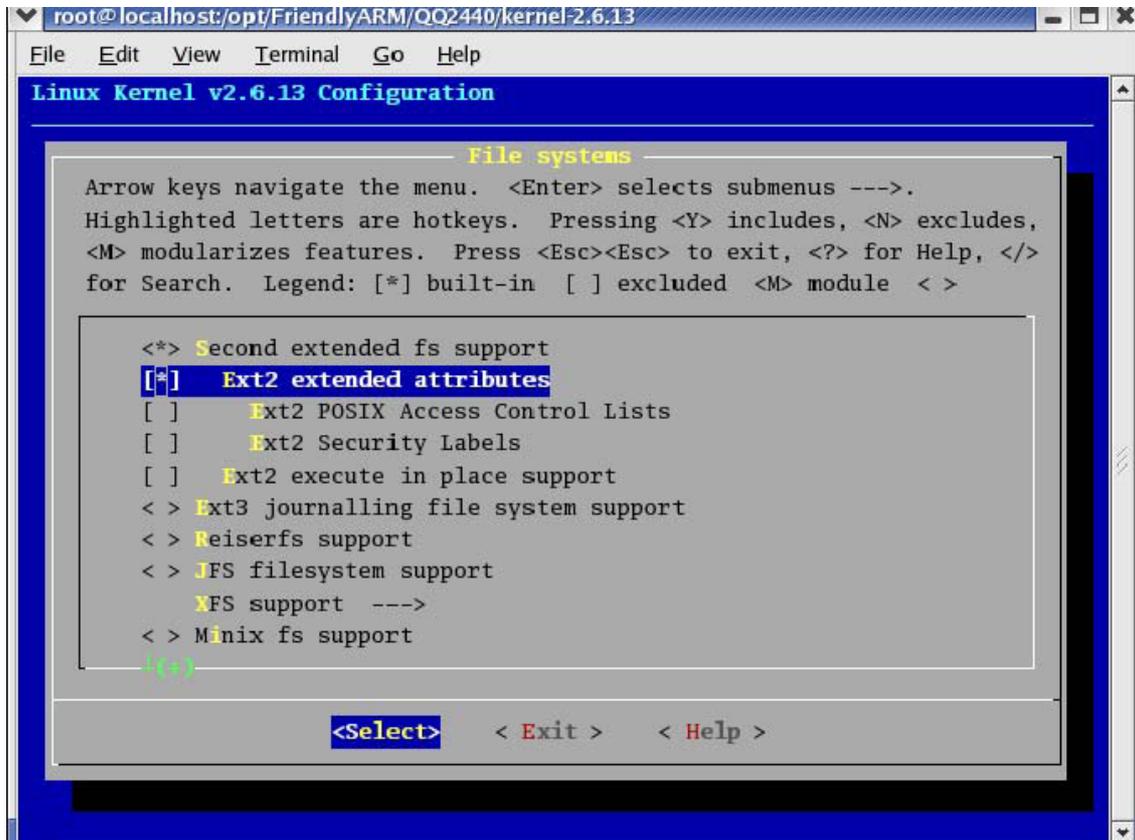
Miscellaneous filesystems
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >

< > EFS file system support (read only) (EXPERIMENTAL)
<*> YAFFS2 file system support
--- 512 byte / page devices
[*] Lets Yaffs do its own ECC
[ ] Use the same ecc byte order as Steven Hill's nand_ecc.c
--- 2048 byte (or larger) / page devices
[*] Turn off debug chunk erase check
[*] Cache short names in RAM
< > Journalling Flash File System (JFFS) support
< > Journalling Flash File System v2 (JFFS2) support

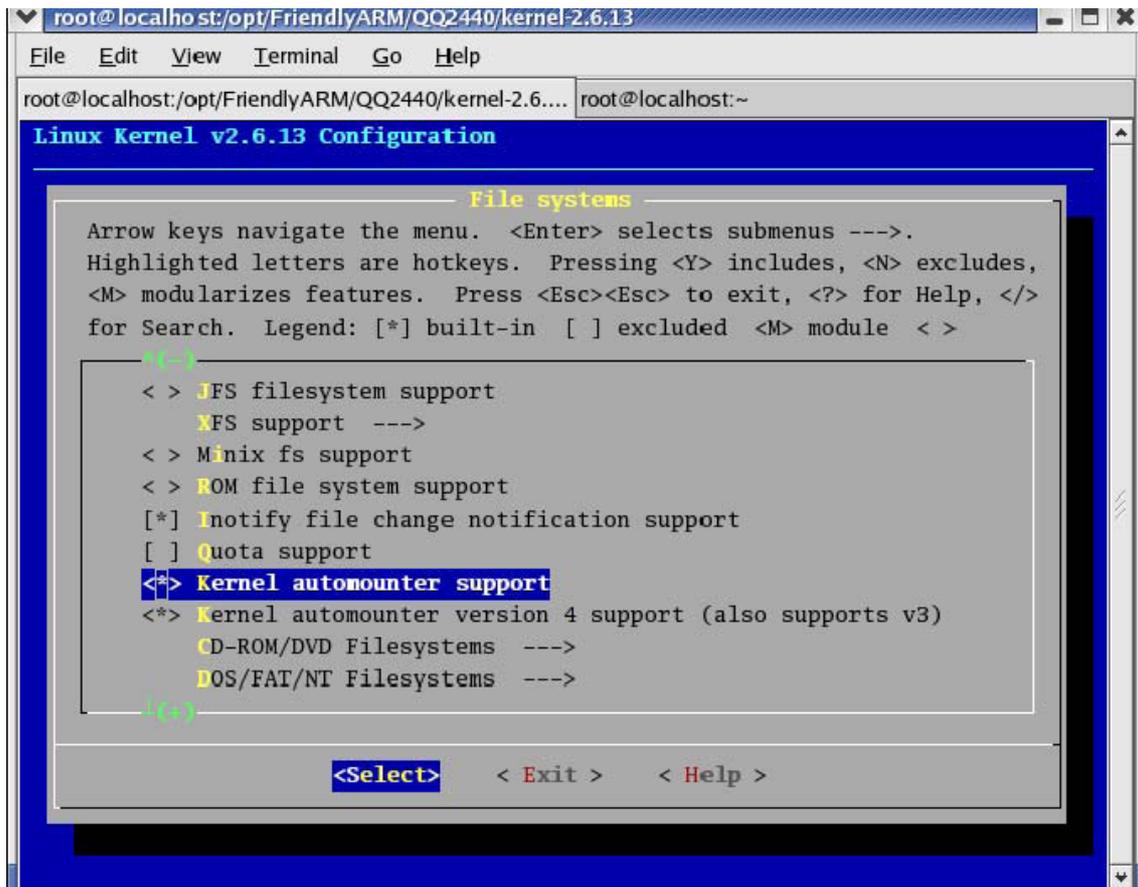
<Select> < Exit > < Help >
```

5.5.14 EXT2/VFAT/ NFS ファイルシステム

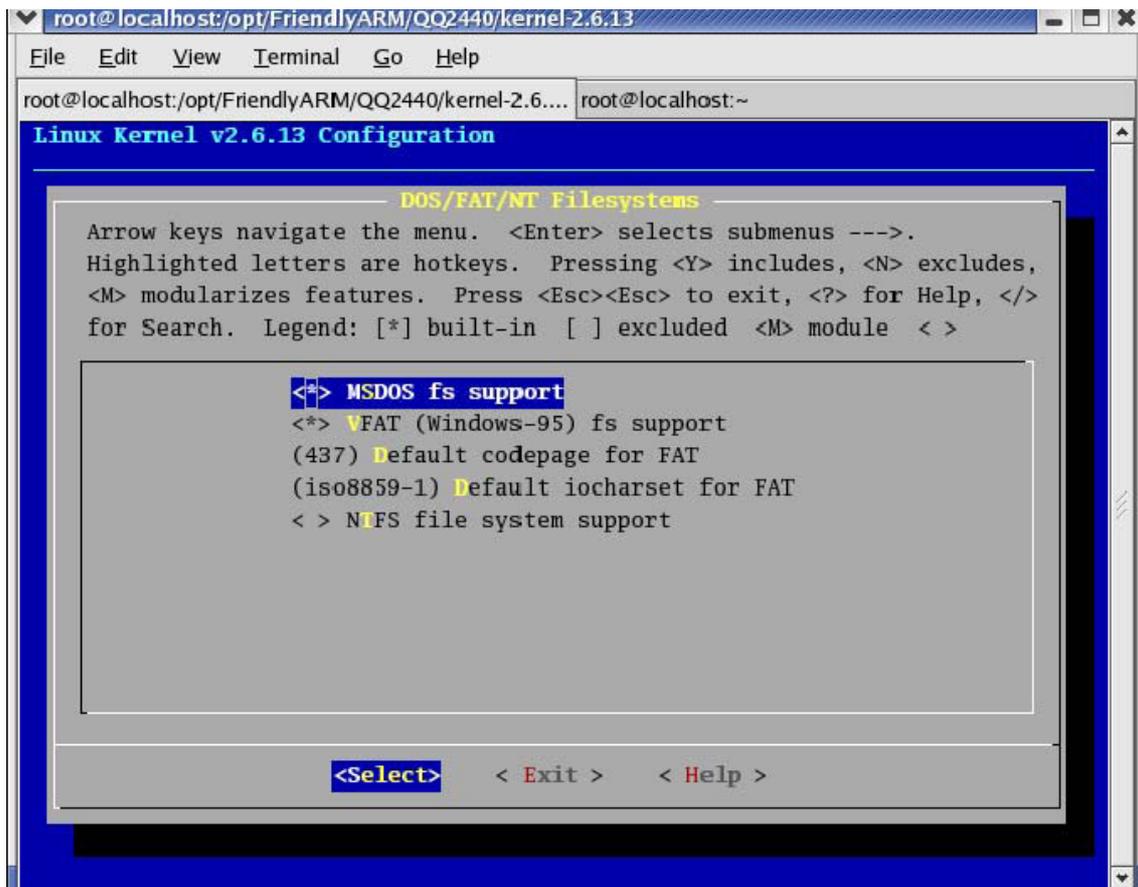
メニュー「File Systems -->」のなかで



自動的なマウントをサポートします。

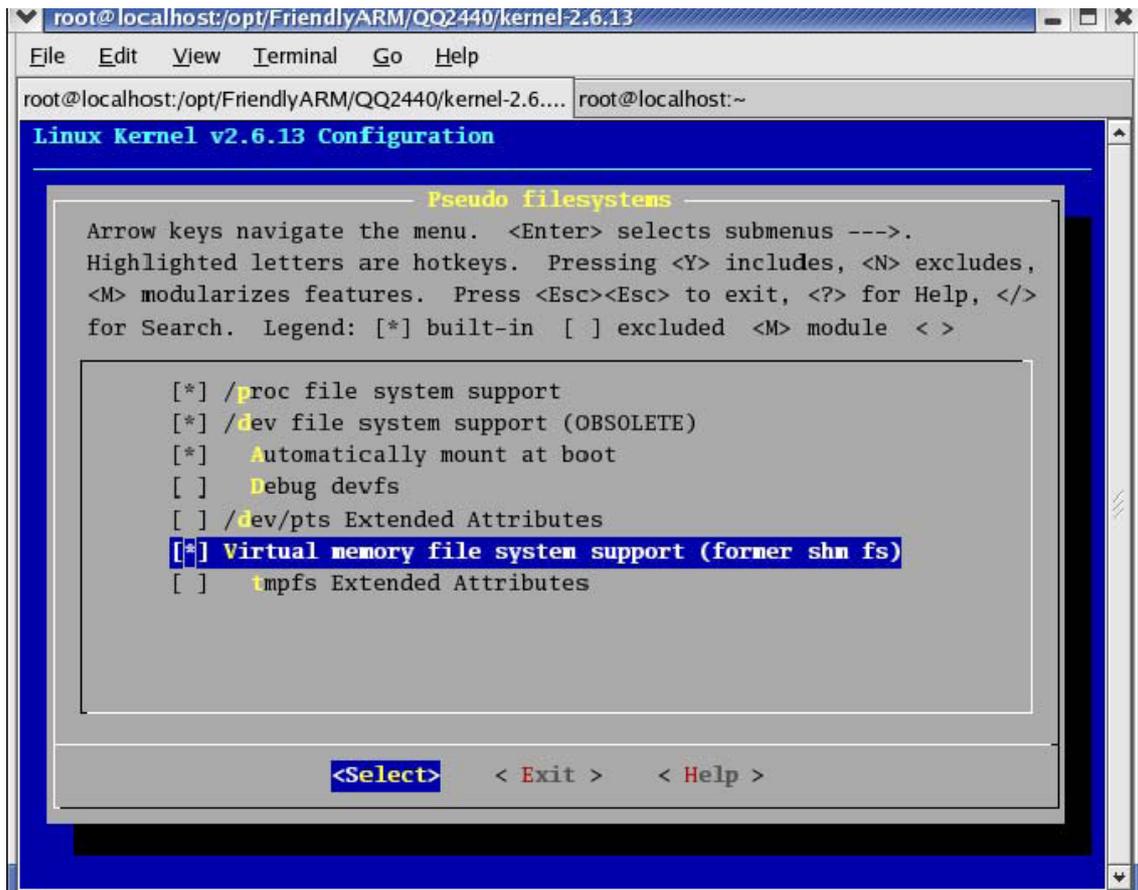


メニュー「DOS/FAT/NT Filesystems」でFATをサポートします。



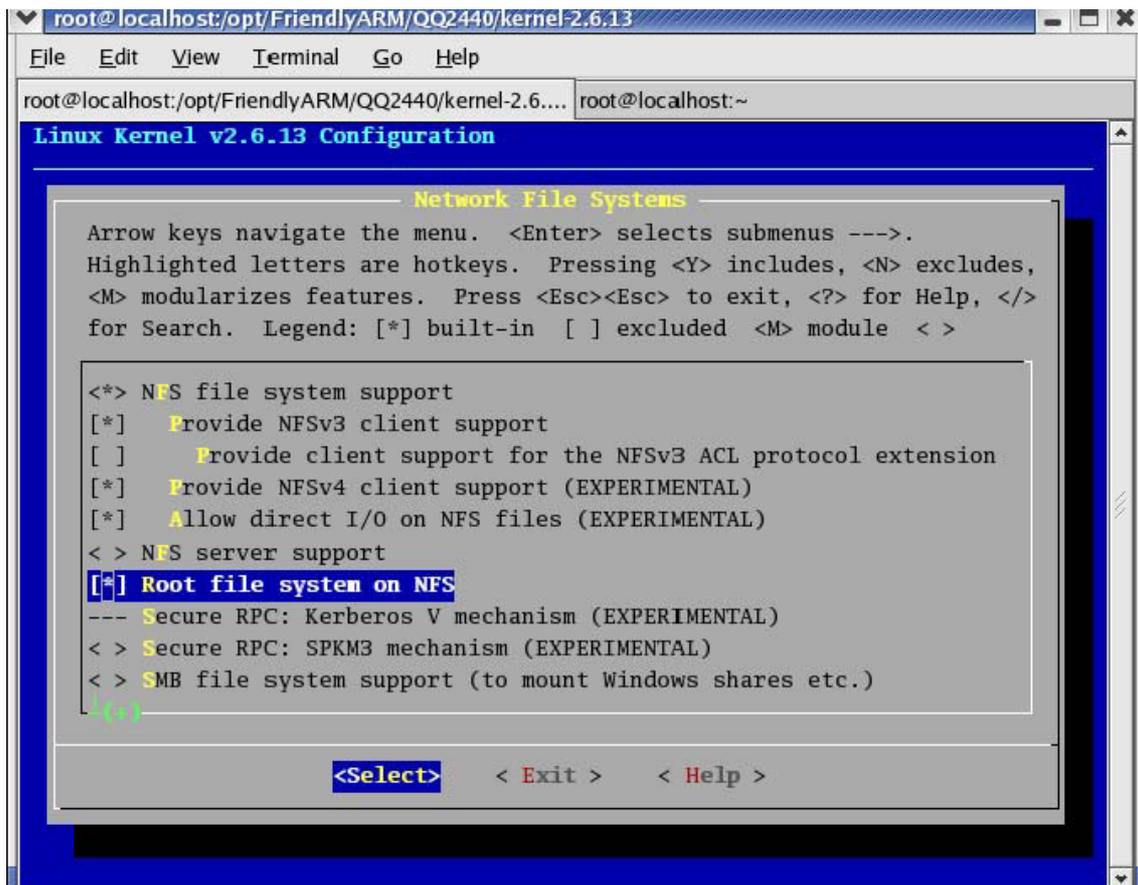
メニュー「File System」→「Pseudo filesystems --->」

これを選択しないと、yaffsファイルシステムが動けません。



メニュー「File System」→「Network File Systems --->」

これを選択しないと、NFSを利用できません。



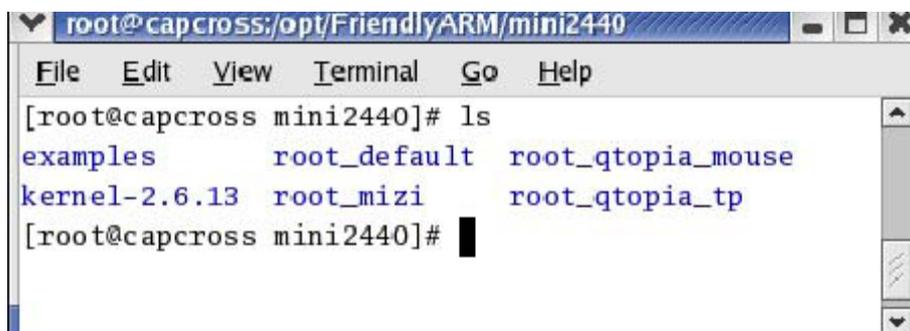
5.6 yaffs ルートファイルシステムのイメージを生成

1. yaffsイメージを生成するツールを解凍します。

```
# tar xvzf mkyaffsimage.tgz -C /usr/sbin
```

2. デフォルトルートファイルシステムを解凍します。

```
# tar xvzf root_default.tgz
```

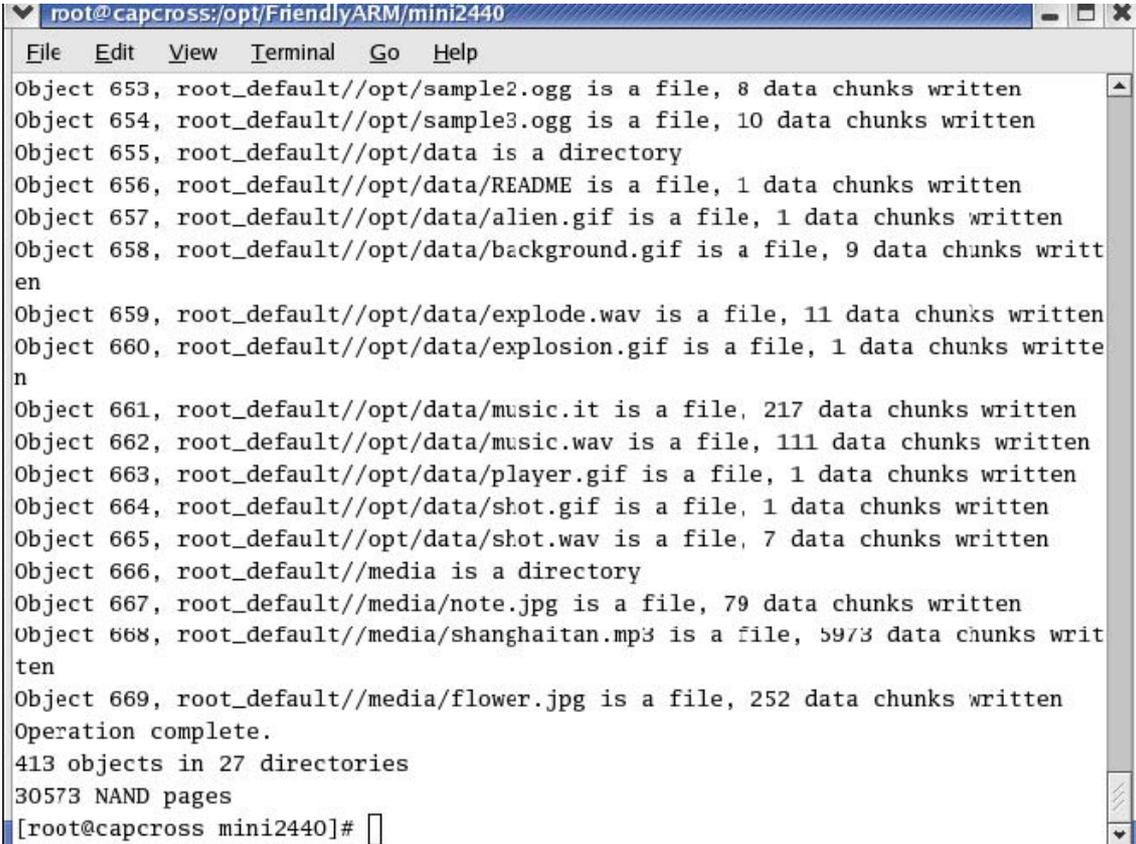


3. 開発された実行ファイルをコピーします

```
# cp examples/hello/hello root_default/sbin
```

4. yaffsイメージを生成します

```
# mkyaffsimage root_default root_default.img
```



```
root@capcross:/opt/FriendlyARM/mini2440
File Edit View Terminal Go Help
Object 653, root_default//opt/sample2.ogg is a file, 8 data chunks written
Object 654, root_default//opt/sample3.ogg is a file, 10 data chunks written
Object 655, root_default//opt/data is a directory
Object 656, root_default//opt/data/README is a file, 1 data chunks written
Object 657, root_default//opt/data/alien.gif is a file, 1 data chunks written
Object 658, root_default//opt/data/background.gif is a file, 9 data chunks written
Object 659, root_default//opt/data/explode.wav is a file, 11 data chunks written
Object 660, root_default//opt/data/explosion.gif is a file, 1 data chunks written
Object 661, root_default//opt/data/music.it is a file, 217 data chunks written
Object 662, root_default//opt/data/music.wav is a file, 111 data chunks written
Object 663, root_default//opt/data/player.gif is a file, 1 data chunks written
Object 664, root_default//opt/data/shot.gif is a file, 1 data chunks written
Object 665, root_default//opt/data/shot.wav is a file, 7 data chunks written
Object 666, root_default//media is a directory
Object 667, root_default//media/note.jpg is a file, 79 data chunks written
Object 668, root_default//media/shanghai.mp3 is a file, 5973 data chunks written
Object 669, root_default//media/flower.jpg is a file, 252 data chunks written
Operation complete.
413 objects in 27 directories
30573 NAND pages
[root@capcross mini2440]#
```

5.7 Linux ドライバの開発入門

Linuxなどの現代的なOSでは、デバイスに対する入出力はデバイスドライバを通じて行うのが常識です。Linuxは「特権モード」を使い、カーネルモードとユーザーモードを厳密に分離しています。ユーザーモードからは、物理メモリアドレスやI/Oポートなどへのアクセスはできません。したがって、デバイスに対する入出力は、カーネルモードで動作するドライバを通じて行うしかありません。

ある例を通じて、カーネルモードで動作するドライバの設計を紹介します。

5.7.1 簡単なドライバのソースコード

ソースコード : kernel-2.6.13/drivers/char/qq2440_hello_sample.c

```
#include <linux/kernel.h>
#include <linux/module.h>
MODULE_LICENSE("GPL");
static int __init qq2440_hello_module_init(void)
{
    printk("Hello, QQ2440 module is installed !\n");
    return 0;
}
static void __exit qq2440_hello_module_cleanup(void)
{
    printk("Good-bye, QQ2440 module was removed!\n");
}
module_init(qq2440_hello_module_init);
module_exit(qq2440_hello_module_cleanup);
```

5.7.2 コンフィグファイルを編集します

kernel-2.6.13/drivers/char/Kconfigを開きます。下の内容を添加します(実は、添加完了しました、確認してみます)。

```
root@localhost:/opt/FriendlyARM/QQ2440/kernel-2.6.13
File Edit View Terminal Go Help
depends on ARCH_S3C2410
help
RTC (Realtime Clock) driver for the clock inbuilt into the
Samsung S3C2410. This can provide periodic interrupt rates
from 1Hz to 64Hz for user programs, and wakeup from Alarm.

config SBC2440_LEDS
tristate "SBC2440 LEDs Driver"
depends on ARCH_S3C2410
help
SBC2440 User leds.

config QQ2440_HELLO_MODULE
tristate "QQ2440 module sample"
depends on ARCH_S3C2410
help
QQ2440 module sample.

config RTC_VR41XX
tristate "NEC VR4100 series Real Time Clock Support"
depends on CPU_VR41XX

config COBALT_LCD

787,0-1 77%
```

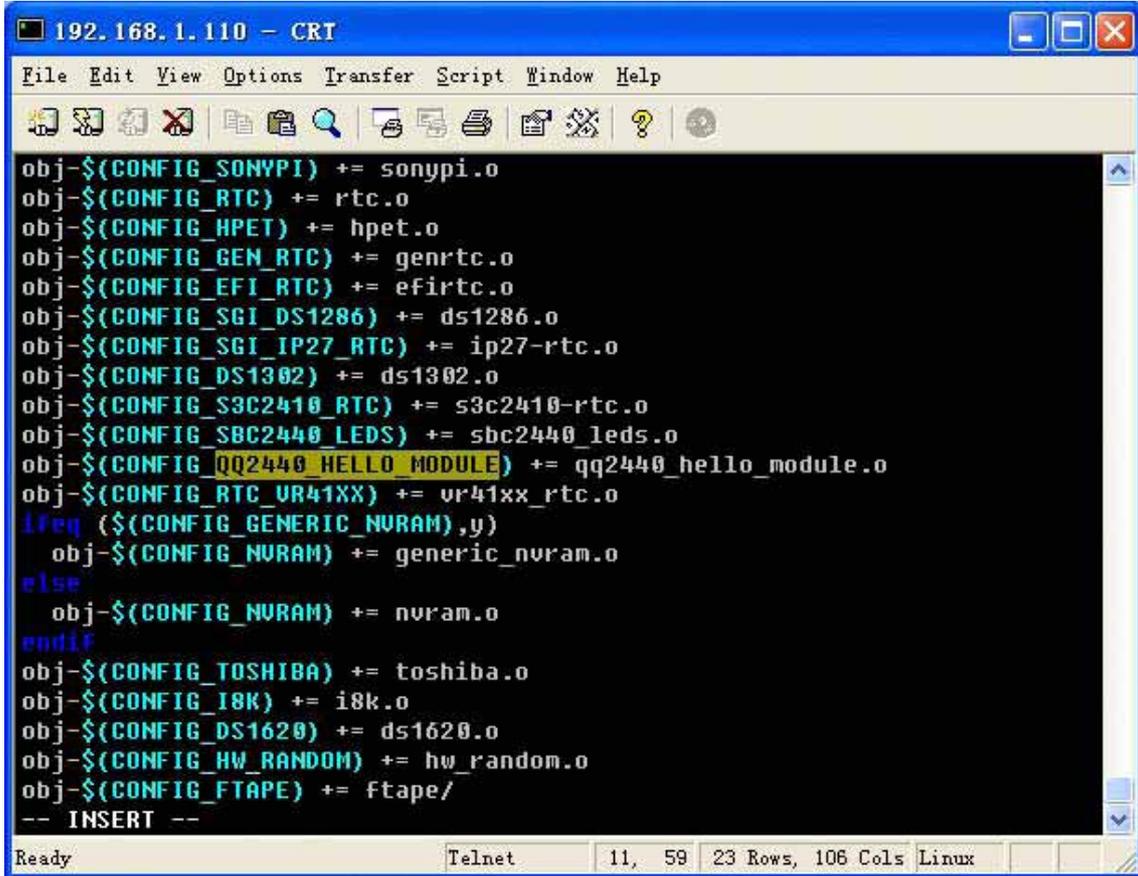
kernel-2.6.13でmake menuconfigを実行して、メニューDevice Drivers → Character devicesを選んで、

```
Linux Kernel v2.6.13 Configuration
-----
Character devices
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are
Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit
for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module c
-----
^(-)
[ ] Legacy (BSD) PTY support
    |PHI --->
    Watchdog Cards --->
< > /dev/nvram support
< > Enhanced Real Time Clock Support
[*] S3C2410 RTC Driver
<*> SBC2440 LEDs Driver
<M> QQ2440 module sample
< > Double Talk PC internal speech card support
< > Siemens R3964 line discipline
    |tape, the floppy tape device driver --->
< > RAW driver (/dev/raw/rawN) (OBSOLETE)
    |TPM devices --->
-----
<Select> < Exit > < Help >
```

追加されたものが見えます。spaceキーで「M」を選択します。

5.7.3 Makefile を編集

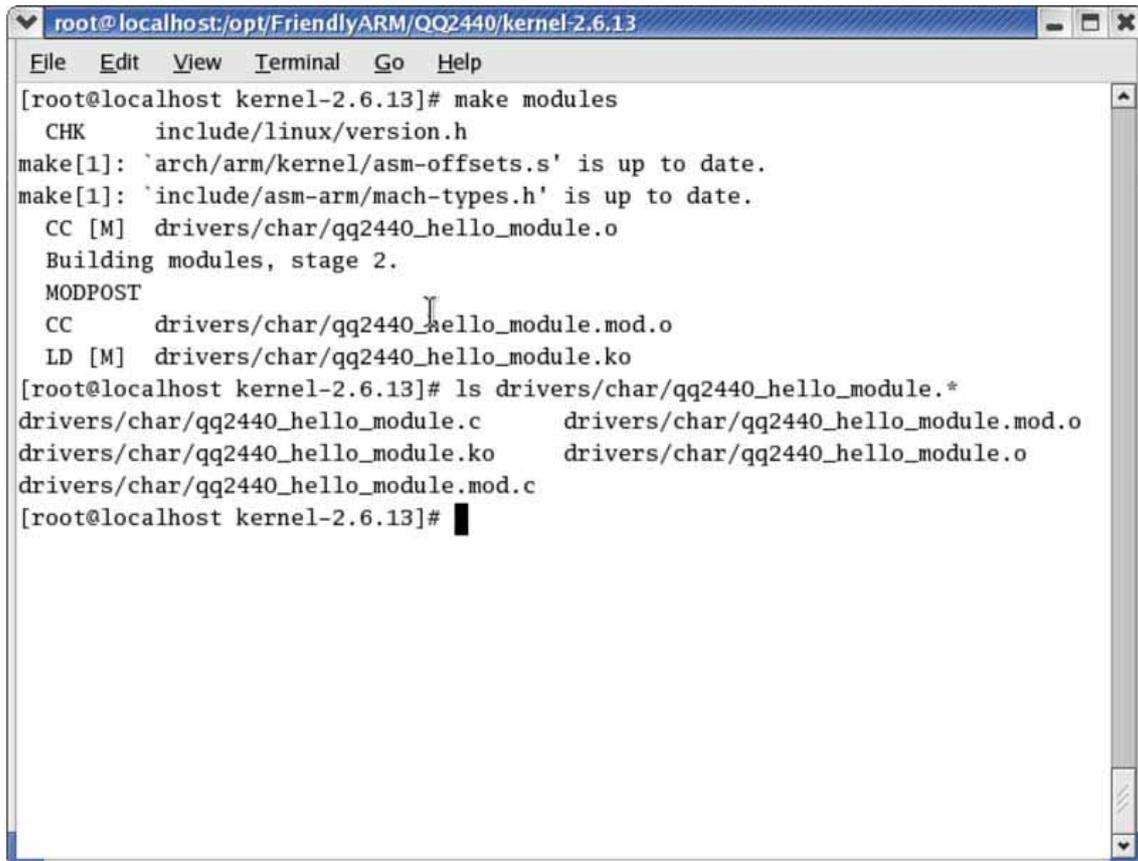
kernel-2.6.13/drivers/char/Makefile を開きます。下の内容を追加します(実は、追加完了しました、確認してみます)。



```
obj-$(CONFIG_SONYPI) += sonypi.o
obj-$(CONFIG_RTC) += rtc.o
obj-$(CONFIG_HPET) += hpet.o
obj-$(CONFIG_GEN_RTC) += genrtc.o
obj-$(CONFIG_EFI_RTC) += efirtc.o
obj-$(CONFIG_SGI_DS1286) += ds1286.o
obj-$(CONFIG_SGI_IP27_RTC) += ip27-rtc.o
obj-$(CONFIG_DS1302) += ds1302.o
obj-$(CONFIG_S3C2410_RTC) += s3c2410-rtc.o
obj-$(CONFIG_SBC2440_LEDS) += sbc2440_leds.o
obj-$(CONFIG_QQ2440_HELLO_MODULE) += qq2440_hello_module.o
obj-$(CONFIG_RTC_VR41XX) += vr41xx_rtc.o
ifeq ($(CONFIG_GENERIC_NVRAM),y)
  obj-$(CONFIG_NVRAM) += generic_nvram.o
else
  obj-$(CONFIG_NVRAM) += nvram.o
endif
obj-$(CONFIG_TOSHIBA) += toshiba.o
obj-$(CONFIG_I8K) += i8k.o
obj-$(CONFIG_DS1620) += ds1620.o
obj-$(CONFIG_HW_RANDOM) += hw_random.o
obj-$(CONFIG_FTAPE) += ftape/
-- INSERT --
```

5.7.4 ドライバをコンパイルします

kernel-2.6.13でmake modulesを実行します。kernel-2.6.13/drivers/char/でオブジェクトファイルqq2440_hello_module.koを生成させます。



```
root@localhost:/opt/FriendlyARM/QQ2440/kernel-2.6.13
File Edit View Terminal Go Help
[root@localhost kernel-2.6.13]# make modules
CHK include/linux/version.h
make[1]: `arch/arm/kernel/asm-offsets.s' is up to date.
make[1]: `include/asm-arm/mach-types.h' is up to date.
CC [M] drivers/char/qq2440_hello_module.o
Building modules, stage 2.
MODPOST
CC drivers/char/qq2440_hello_module.mod.o
LD [M] drivers/char/qq2440_hello_module.ko
[root@localhost kernel-2.6.13]# ls drivers/char/qq2440_hello_module.*
drivers/char/qq2440_hello_module.c      drivers/char/qq2440_hello_module.mod.o
drivers/char/qq2440_hello_module.ko     drivers/char/qq2440_hello_module.o
drivers/char/qq2440_hello_module.mod.c
[root@localhost kernel-2.6.13]#
```

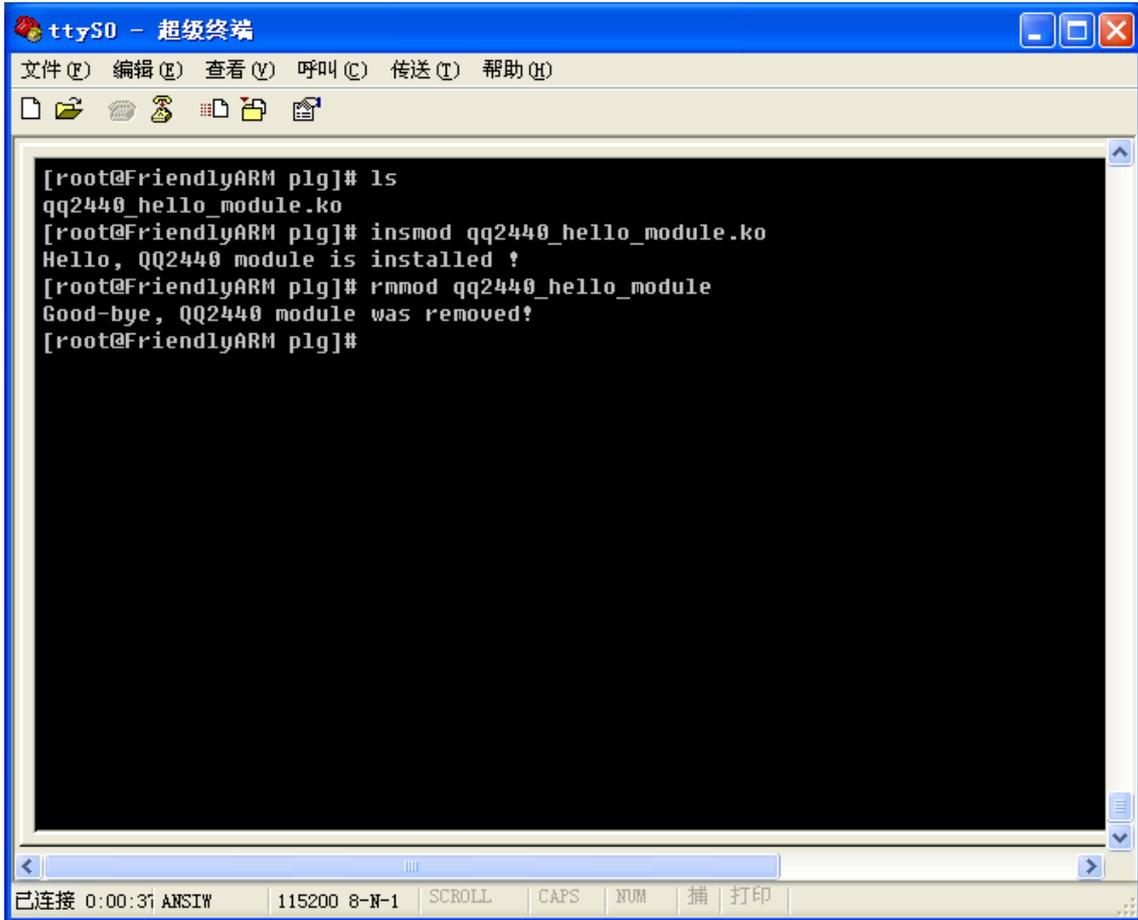
5.7.5 ARM9 ボードでドライバをインストールします

ドライバをロードします。

```
#insmod qq2440_hello_module.ko
```

ドライバを削除します。

```
#rmmod qq2440_hello_module.ko
```



The image shows a terminal window titled "ttyS0 - 超級终端". The window contains the following text:

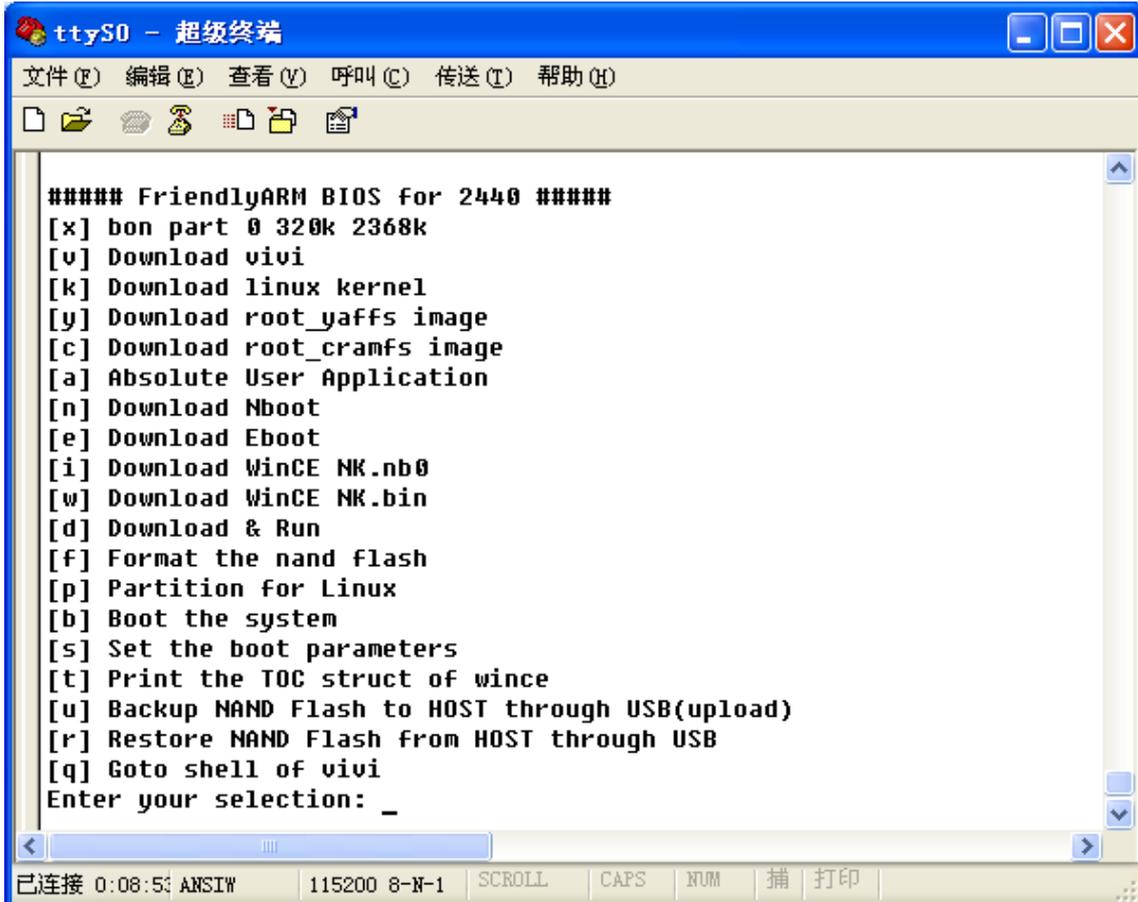
```
[root@FriendlyARM plg]# ls
qq2440_hello_module.ko
[root@FriendlyARM plg]# insmod qq2440_hello_module.ko
Hello, QQ2440 module is installed !
[root@FriendlyARM plg]# rmmod qq2440_hello_module
Good-bye, QQ2440 module was removed!
[root@FriendlyARM plg]#
```

The terminal window also shows a menu bar with options: 文件(F), 编辑(E), 查看(V), 呼叫(C), 传送(T), 帮助(H). At the bottom, there is a status bar with the text: 已连接 0:00:31 ANSIV 115200 8-N-1 SCROLL CAPS NUM 捕 打印.

第六章 生成されたファイルを書き込む

6.1 NOR Flash から起動

ARM9ボードのジャンパーJ1をNor Flashに設定して、電源を入れて、ARM9ボードはNor Flashから起動します。



```
##### FriendlyARM BIOS for 2440 #####
[x] bon part 0 320k 2368k
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[c] Download root_cramfs image
[a] Absolute User Application
[n] Download Nboot
[e] Download Eboot
[i] Download WinCE NK.nb0
[w] Download WinCE NK.bin
[d] Download & Run
[f] Format the nand flash
[p] Partition for Linux
[b] Boot the system
[s] Set the boot parameters
[t] Print the TOC struct of wince
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
Enter your selection: _
```

6.2 USB ドライバのインストール

開発されたOSとプログラムをUSB通じてQQ2240v3にダウンロードします。その為、USBケーブルでQQ2240v3のUSBスレーブポートとパソコンのUSBポートを繋ぐことが必要です。繋ぐと、パソコンは新しいデバイスを発見して、USBドライバをインストールします。

新しいハードウェアの検出ウィザード



新しいハードウェアの検索ウィザードの開始

お使いのコンピュータ、ハードウェアのインストール CD または Windows Update の Web サイトを検索して (ユーザーの了解のもとに) 現在のソフトウェアおよび更新されたソフトウェアを検索します。
[プライバシー ポリシー](#)を表示します。

ソフトウェア検索のため、Windows Update に接続しますか?

- はい、今回のみ接続します (Y)
- はい、今すぐおよびデバイスの接続時には毎回接続します (E)
- いいえ、今回は接続しません (N)

続行するには、[次へ] をクリックしてください。

< 戻る (B)

次へ (N) >

キャンセル

新しいハードウェアの検出ウィザード



このウィザードでは、次のハードウェアに必要なソフトウェアをインストールします:
SEC S302410X Test B/D



ハードウェアに付属のインストール CD またはフロッピー ディスクがある場合は、挿入してください。

インストール方法を選んでください。

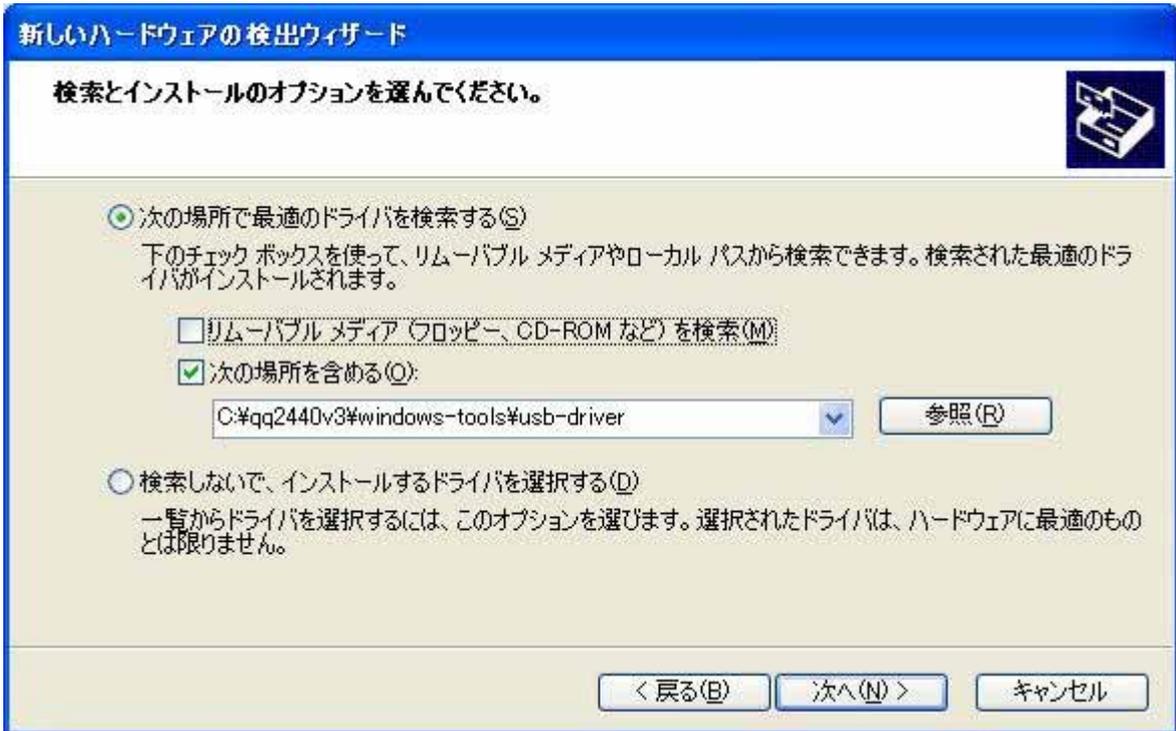
- ソフトウェアを自動的にインストールする (推奨) (Y)
- 一覧または特定の場所からインストールする (詳細) (S)

続行するには、[次へ] をクリックしてください。

< 戻る (B)

次へ (N) >

キャンセル





USBドライバをインストール完了あと、パソコンのダウンロード・ツールDNW.exeを実行して、QQ2240v3とパソコンを繋ぐことが確認できます。

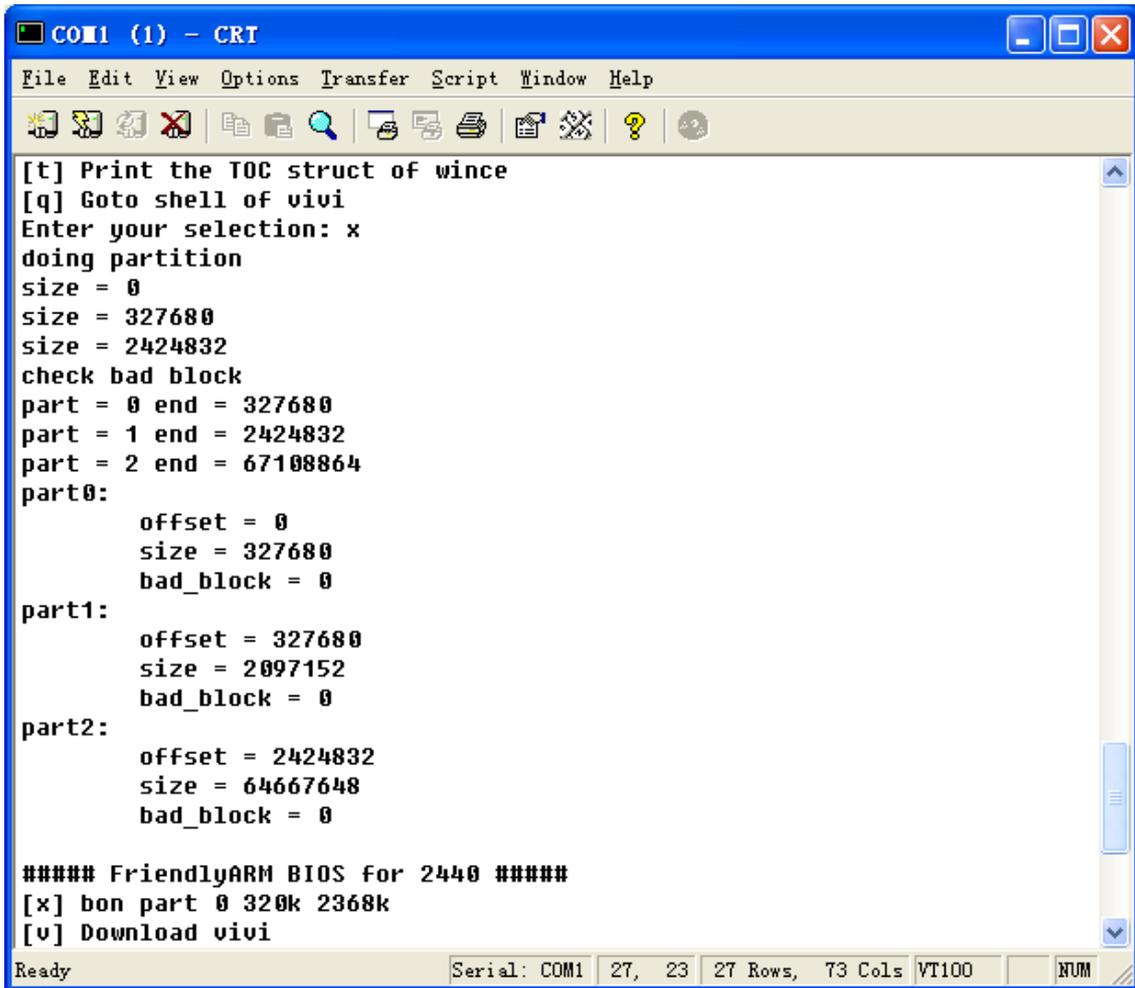


USBドライバはバグがあります。ARM9ボードが再起動、又はリセットの時、ホスト側は死んだかもしれません。その原因で、ARM9ボードが起動完了した後、USBケーブルでホストを繋ぎます。

6.3 NAND Flash のパーティション

メニューの中で、機能号[x]を選択して、NAND Flash のパーティション画面が出てきます。

NAND Flash の中にエラーエリアがあるかもしれません。使用の影響がありません。



```
COM1 (1) - CRT
File Edit View Options Transfer Script Window Help
[t] Print the TOC struct of wince
[q] Goto shell of vivi
Enter your selection: x
doing partition
size = 0
size = 327680
size = 2424832
check bad block
part = 0 end = 327680
part = 1 end = 2424832
part = 2 end = 67108864
part0:
  offset = 0
  size = 327680
  bad_block = 0
part1:
  offset = 327680
  size = 2097152
  bad_block = 0
part2:
  offset = 2424832
  size = 64667648
  bad_block = 0

##### FriendlyARM BIOS for 2440 #####
[x] bon part 0 320k 2368k
[v] Download vivi
Ready Serial: COM1 27, 23 27 Rows, 73 Cols VT100 NUM
```

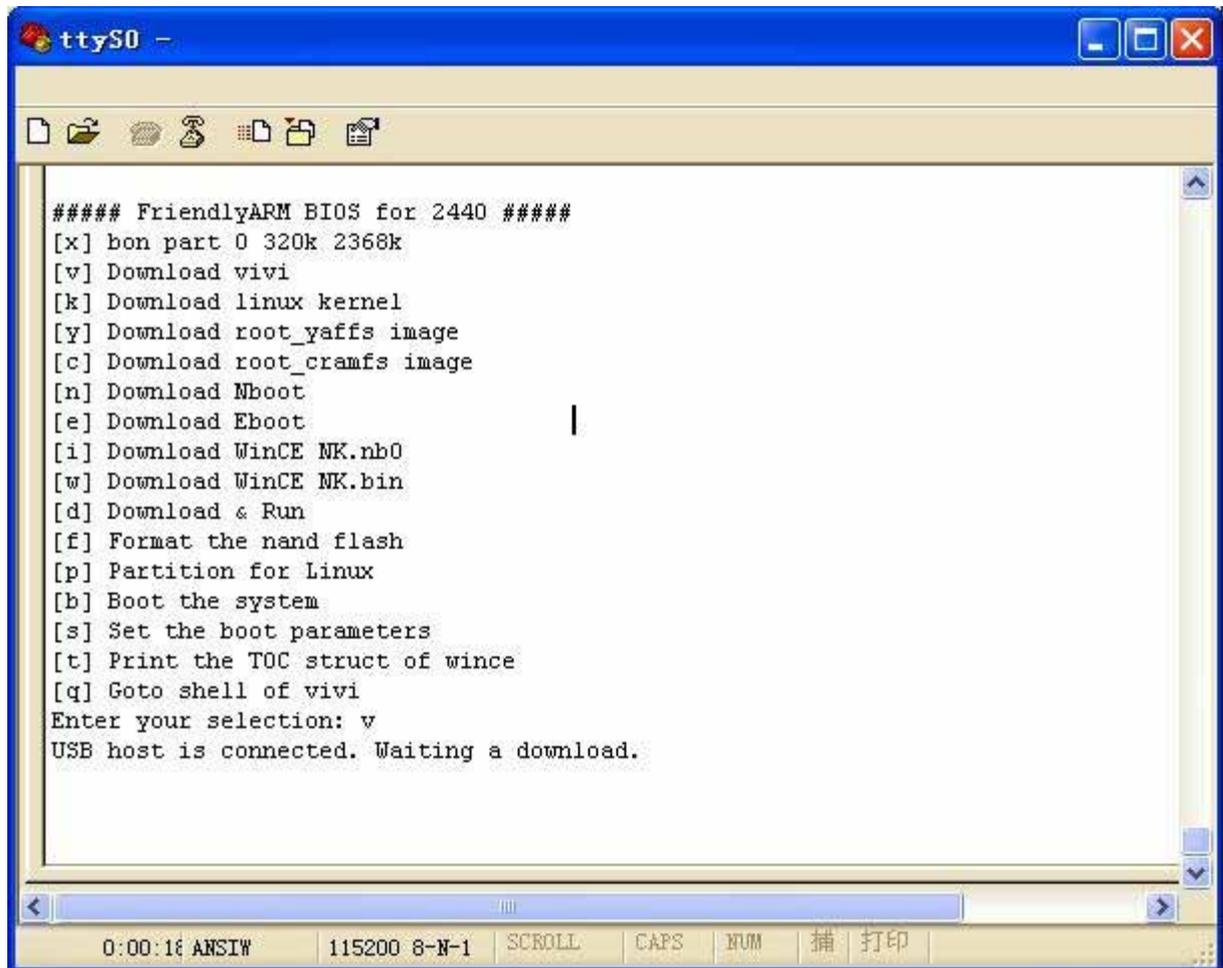
6.4 ブートロードの書き込み

メニューの中で、機能号[x]を選択して、NAND Flash のパーティション画面が出てきます。
パソコンでDNW を実行します。

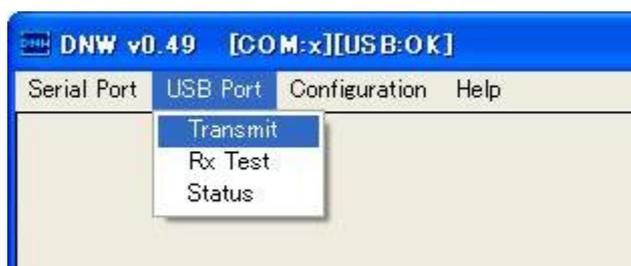


DNW のタイトルに[USB: OK]があれば、パソコンと ARM9 ボードを USB で繋がりました。

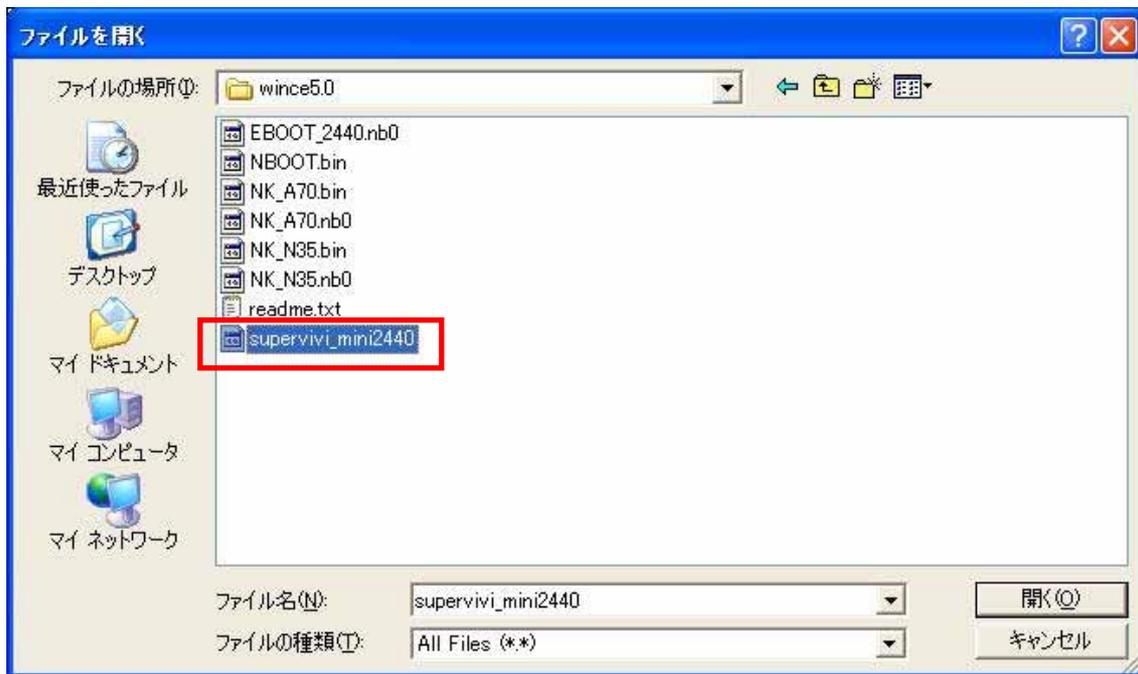
メニューの中で、機能号[v]を選択して、



DNW を待っています。DNW のメニュー「USB Port」→「Transmit」を選択して、



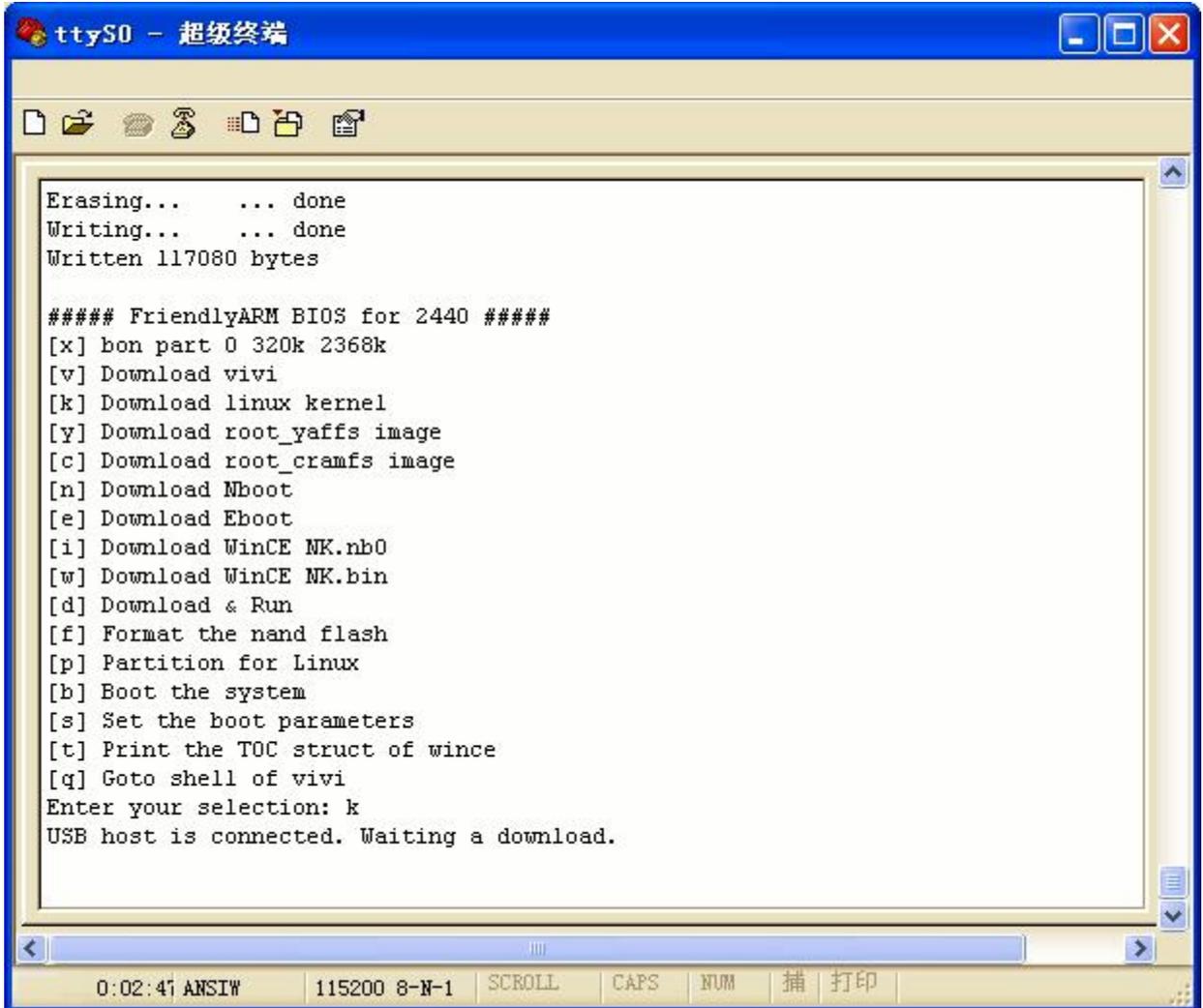
ブートロード supervivi を選択します。



ブートロードを書き込み完了すると、自動的にメニューに戻ります。

6.5 Linux のカーネルの書き込み

a. メニューの中で、機能号[k]を選択して、



```
Erasing...    ... done
Writing...    ... done
Written 117080 bytes

##### FriendlyARM BIOS for 2440 #####
[x] bon part 0 320k 2368k
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[c] Download root_cramfs image
[n] Download Mboot
[e] Download Eboot
[i] Download WinCE NK.nb0
[w] Download WinCE NK.bin
[d] Download & Run
[f] Format the nand flash
[p] Partition for Linux
[b] Boot the system
[s] Set the boot parameters
[t] Print the TOC struct of wince
[q] Goto shell of vivi
Enter your selection: k
USB host is connected. Waiting a download.
```

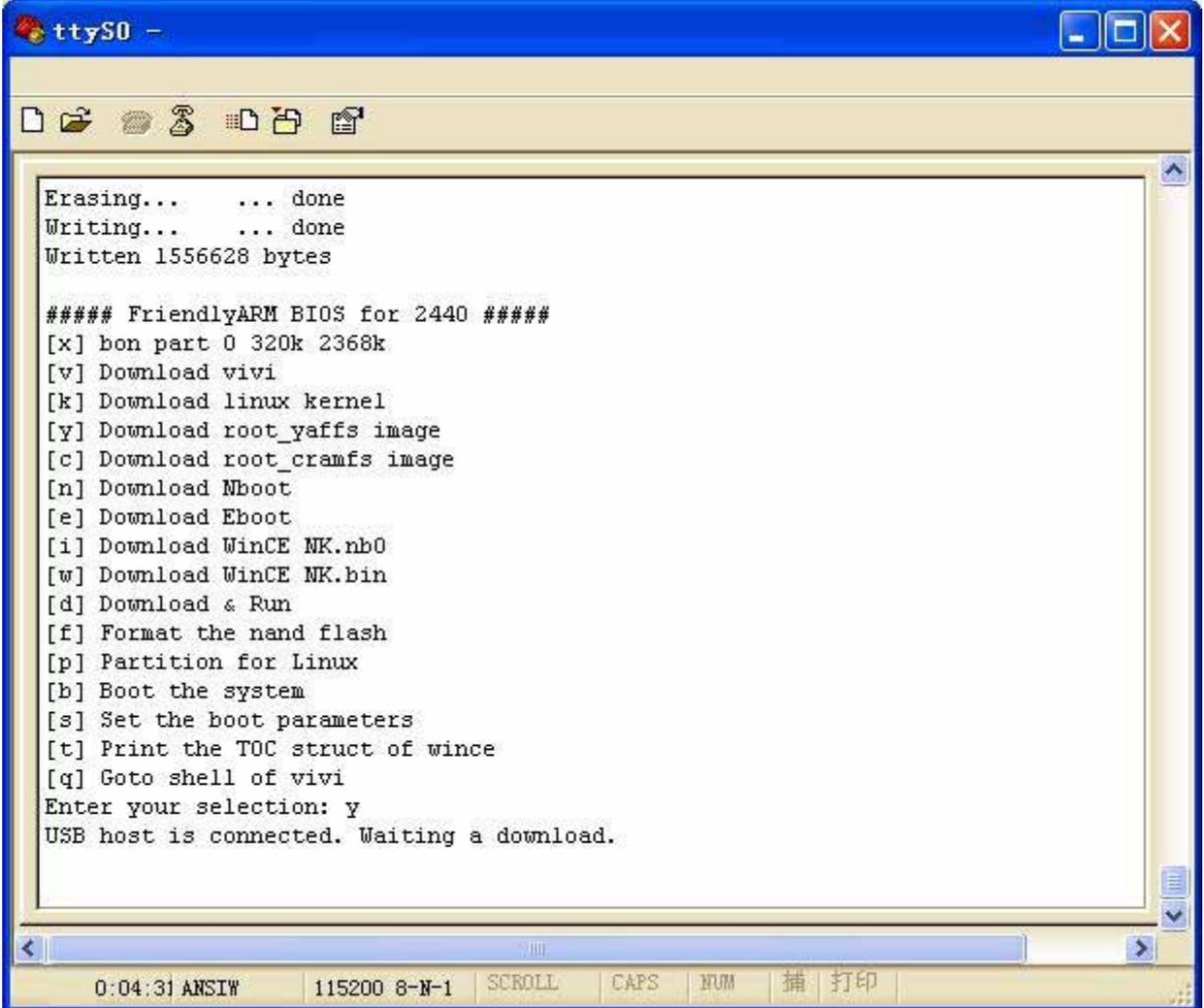
カーネルをダウンロードすることを待っています。

b. DNW のメニュー"USB Port → Transmit"を選択して、カーネルファイル zImage を転送します

c. 転送完了したら、自動的にメニューに戻ります。

6.6 ルート・ファイルシステムの書き込み

a. メニューの中で、機能号[y]を選択して、



```
Erasing...    ... done
Writing...    ... done
Written 1556628 bytes

##### FriendlyARM BIOS for 2440 #####
[x] bon part 0 320k 2368k
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[c] Download root_cramfs image
[n] Download Nboot
[e] Download Eboot
[i] Download WinCE NK.nb0
[w] Download WinCE NK.bin
[d] Download & Run
[f] Format the nand flash
[p] Partition for Linux
[b] Boot the system
[s] Set the boot parameters
[t] Print the TOC struct of wince
[q] Goto shell of vivi
Enter your selection: y
USB host is connected. Waiting a download.
```

ルート・ファイルシステムをダウンロードすることを待っています。

b. DNW のメニュー"USB Port → Transmit"を選択して、ルート・ファイルシステム root_default.img を転送します

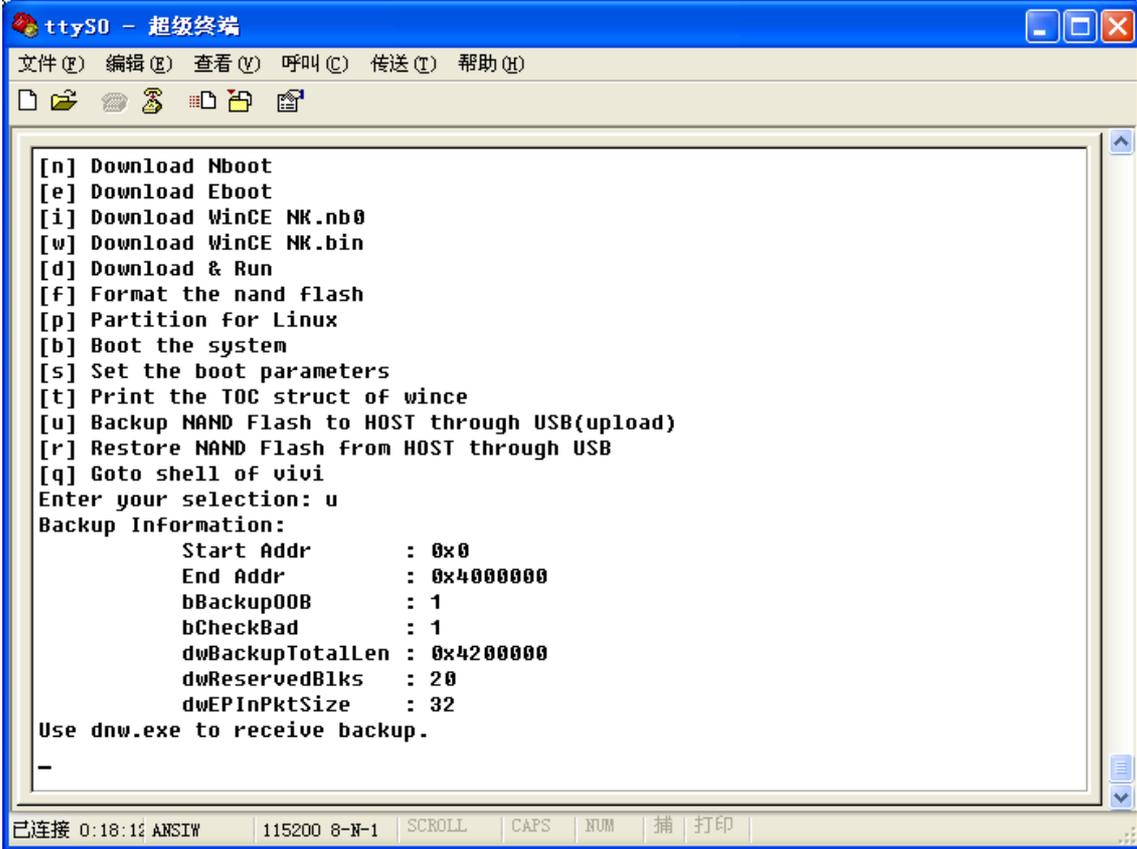
c. 転送完了したら、自動的にメニューに戻ります。

電源を切って、QQ2440v3 の起動 J1 を NAND Flash で起動に設定してください。再び電源を入れて、NAND Flash で書き込み済みの Linux は起動します。

6.7 NAND Flash のバックアップ

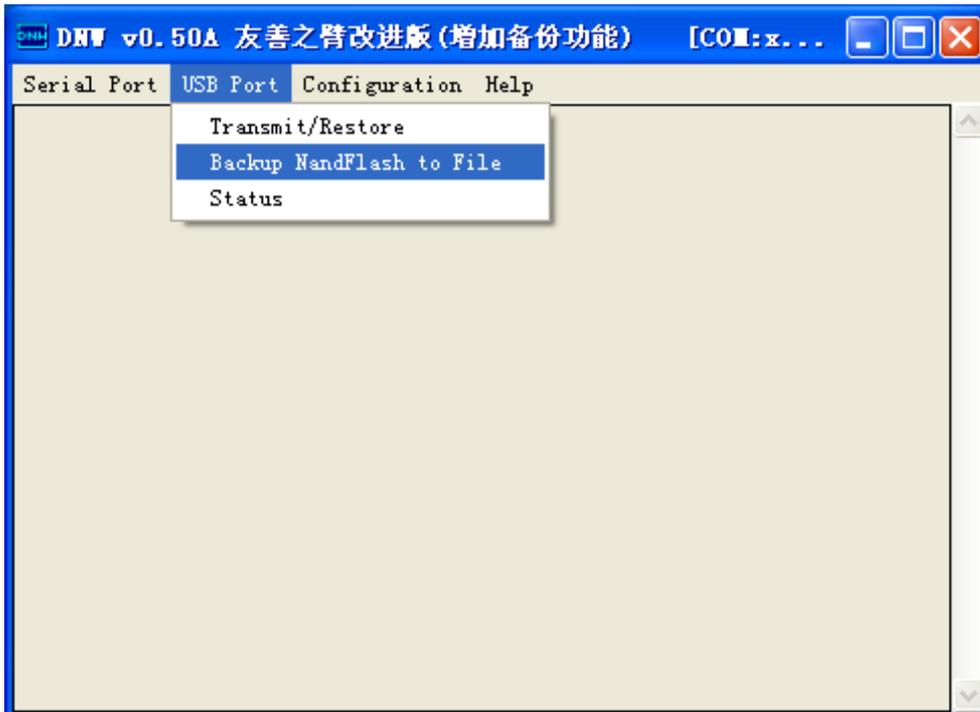
新ブートロード supervivi のみ

メニューの中で、機能号[u]を選択して、

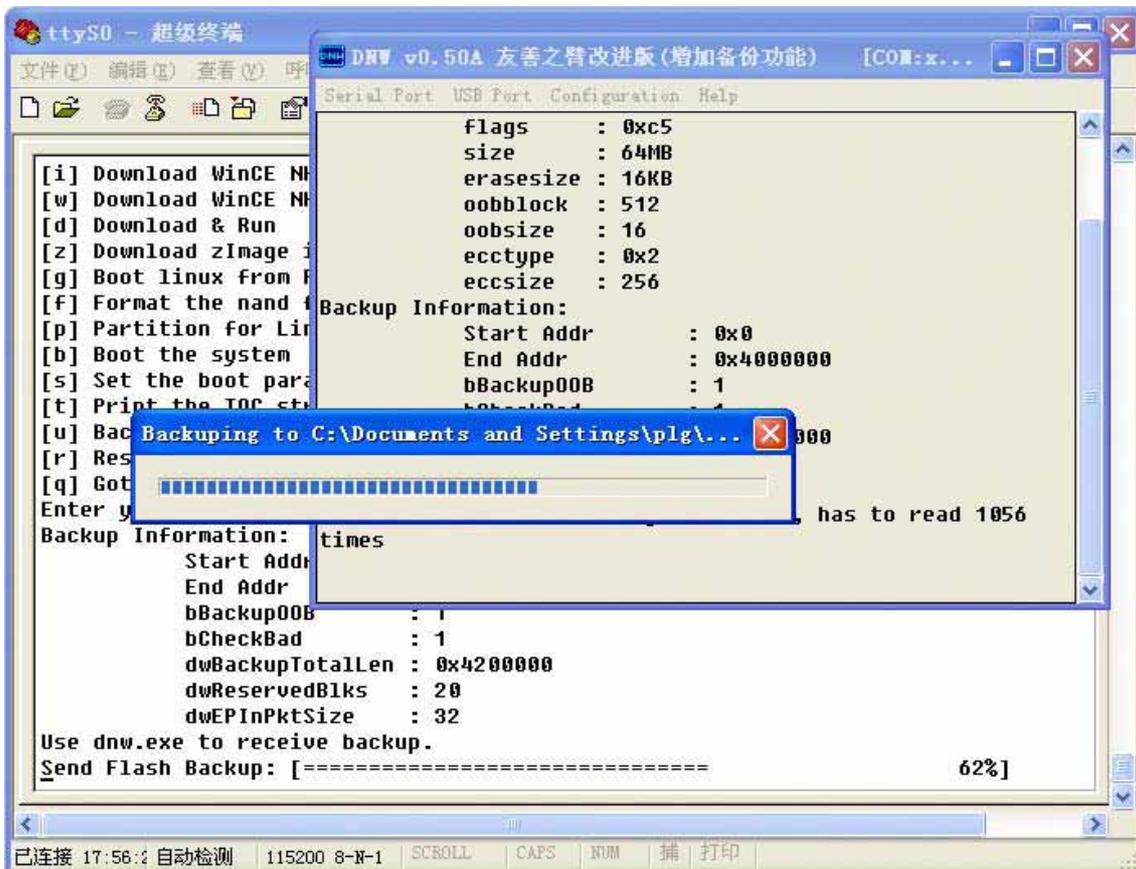


```
ttyS0 - 超級终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
[n] Download Nboot
[e] Download Eboot
[i] Download WinCE NK.nb0
[w] Download WinCE NK.bin
[d] Download & Run
[f] Format the nand flash
[p] Partition for Linux
[b] Boot the system
[s] Set the boot parameters
[t] Print the TOC struct of wince
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
Enter your selection: u
Backup Information:
      Start Addr      : 0x0
      End Addr        : 0x4000000
      bBackup00B      : 1
      bCheckBad       : 1
      dwBackupTotalLen : 0x4200000
      dwReservedBlks  : 20
      dwEPInPktSize   : 32
Use dnw.exe to receive backup.
-
```

DNW のメニュー「Usb Port」→「Backup NandFlash to File」を選択します。



バックアップのファイルの名前「backup.bin」を入力して



バックアップ完了したら、次の画面：



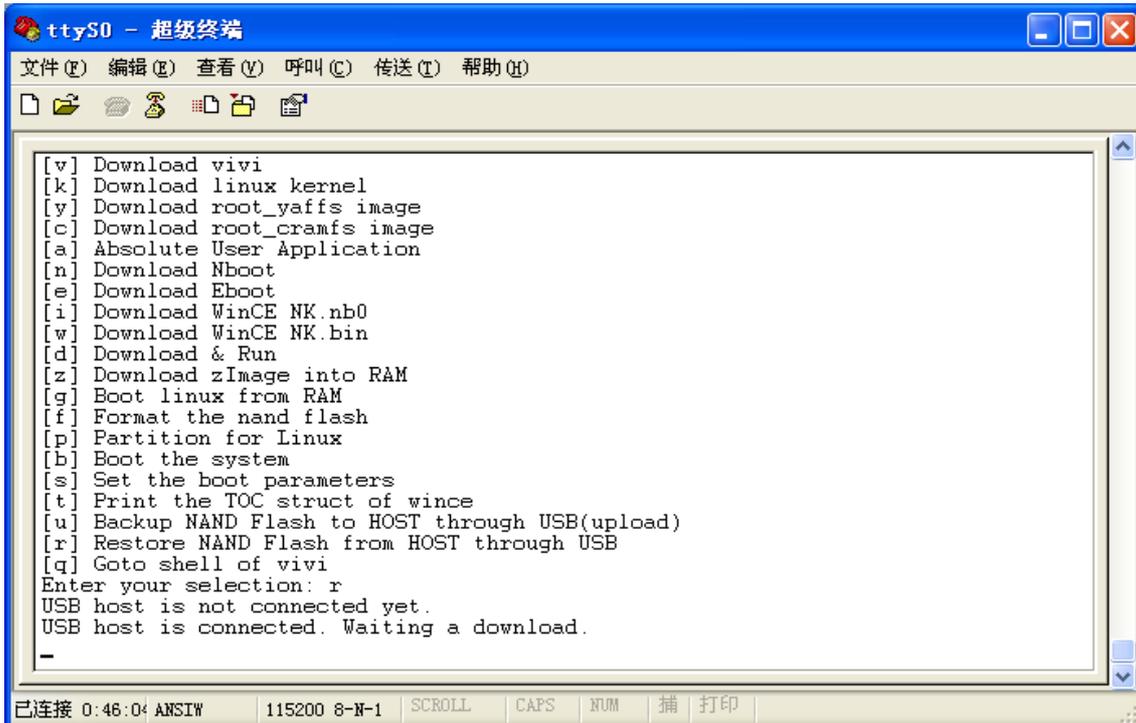
```
Serial Port USB Port Configuration Help
===== USB Backup Start =====
Nand Flash Information:
    type      : 0x4
    flags     : 0xc5
    size      : 64MB
    erasesize : 16KB
    oobblock  : 512
    oobsize   : 16
    ecctype   : 0x2
    eccsize   : 256
Backup Information:
    Start Addr      : 0x0
    End Addr        : 0x4000000
    bBackup00B     : 1
    bCheckBad      : 1
    dwBackupTotalLen : 0x4200000
    dwReservedBlks : 20
    dwEPInPktSize  : 32
dnw.exe read data 65536 bytes a time, has to read 1056
times
===== USB Backup End =====
```

生成されたバックアップファイルの大きさは 66MB ぐらいです。

6.8 NAND Flash のリストア

新ブートロード *supervivi* のみ

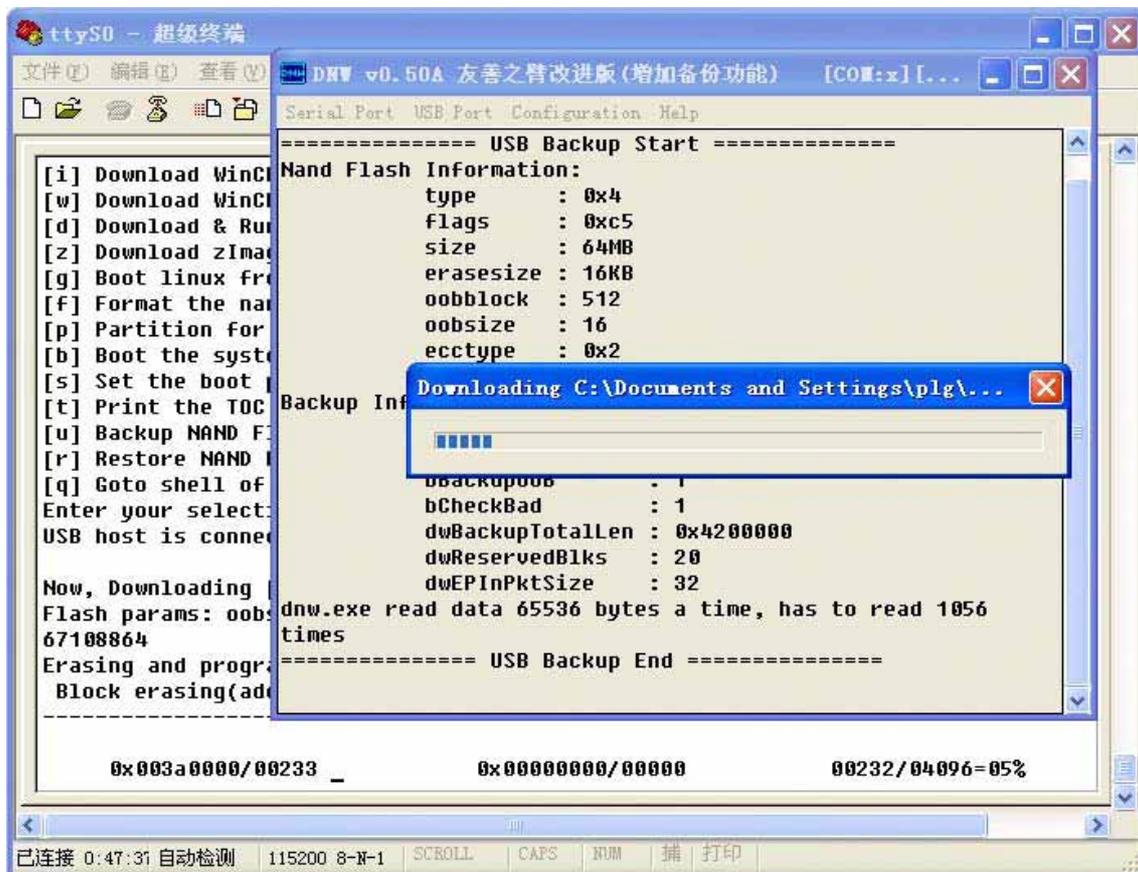
メニューの中で、機能号[r]を選択して、



DNW のメニュー「Usb Port」→「Transmit/Restore」を選択します。



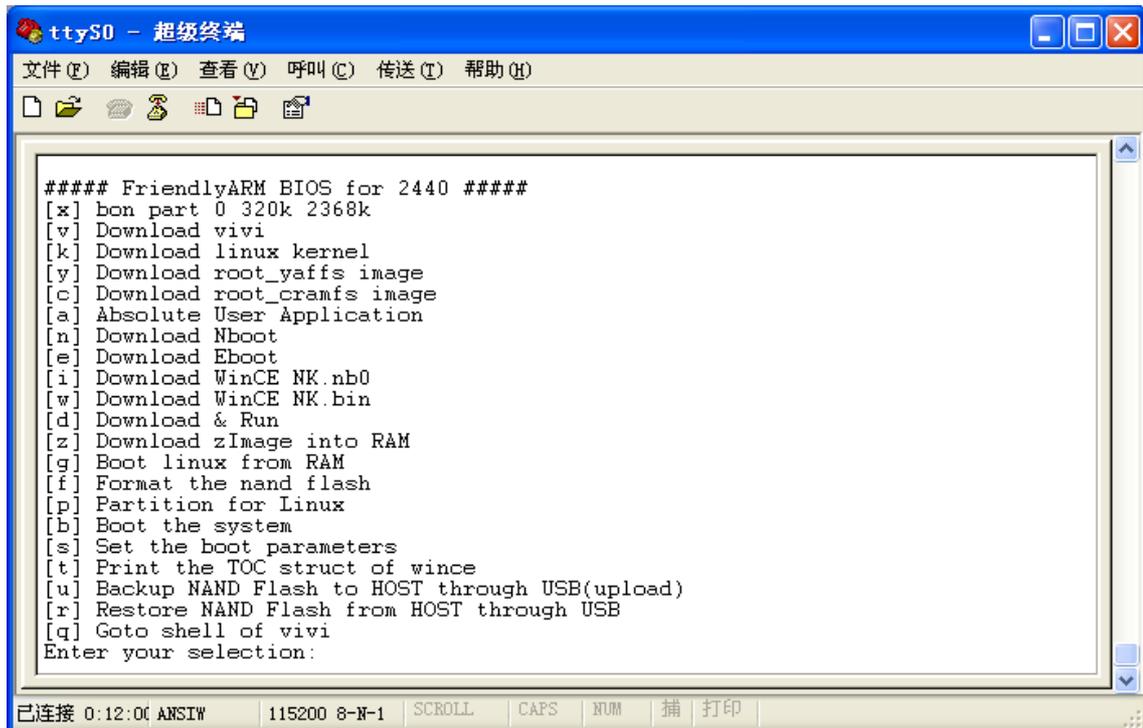
バックアップされたファイル「backup.bin」を選択します。



6.9 メモリで Linux カーネルを直接に実行

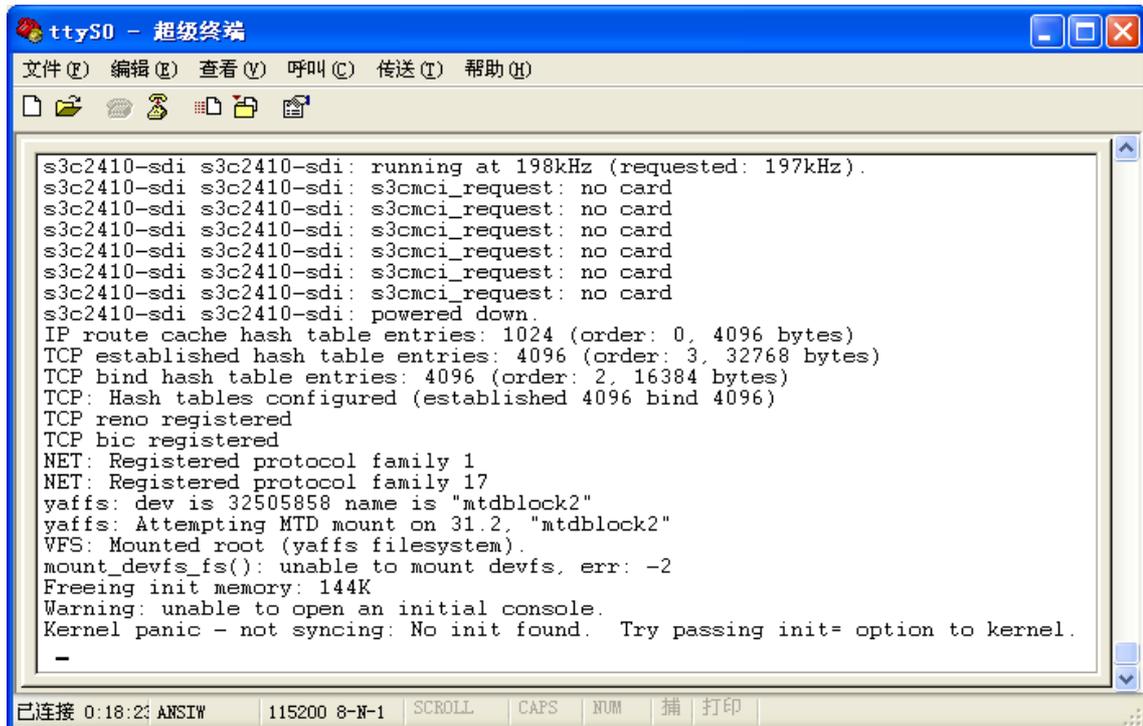
一般的に Linux のカーネルを NAND Flash に書き込み、実行させます。毎回 Linux カーネルを更新すれば、NAND Flash も更新する必要があります。デバッグの時、不便です。ブートロード Supervivi は Linux カーネルをメモリにロードして、直接に実行します。

1. 電源を切って、QQ2440v3 の起動 J1 を Nor Flash で起動に設定してください。再び電源を入れて、Nor Flash で起動します。
2. Supervivi のメニューの中で、機能号[z]を選択して、



```
##### FriendlyARM BIOS for 2440 #####
[x] bon part 0 320k 2368k
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[c] Download root_cramfs image
[a] Absolute User Application
[n] Download Nboot
[e] Download Eboot
[i] Download WinCE NK.nb0
[w] Download WinCE NK.bin
[d] Download & Run
[z] Download zImage into RAM
[g] Boot linux from RAM
[f] Format the nand flash
[p] Partition for Linux
[b] Boot the system
[s] Set the boot parameters
[t] Print the TOC struct of wince
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
Enter your selection:
```

3. DNW のメニュー-"USB Port → Transmit"を選択して、カーネルファイル zImage を転送します
4. 転送完了したら、自動的に Supervivi のメニューに戻ります。機能号[g]を選択して、linux カーネルを実行させます。この画面が出たら：



```
s3c2410-sdi s3c2410-sdi: running at 198kHz (requested: 197kHz).
s3c2410-sdi s3c2410-sdi: s3cmci_request: no card
s3c2410-sdi s3c2410-sdi: powered down.
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 4096 (order: 3, 32768 bytes)
TCP bind hash table entries: 4096 (order: 2, 16384 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP reno registered
TCP bic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
VFS: Mounted root (yaffs filesystem).
mount_devfs_fs(): unable to mount devfs, err: -2
Freeing init memory: 144K
Warning: unable to open an initial console.
Kernel panic - not syncing: No init found. Try passing init= option to kernel.
-
```



ルートファイルシステムが見つかりませんでした！

Supervivi のメニューの中で、機能号[y]を選択して、NAND Flash に root_default.img を書き込みます。ルートファイルシステムを作ります。又は、NFS をルートファイルシステムとして指定します。

Linux カーネルを実行させる前に、NFS を指定します。Supervivi のメニューの中で、機能号[q]を選択して、次のコマンドを入力してください。

```
Supervivi>param set linux_cmd_line "console=ttySAC0 root=/dev/nfs
nfsroot=192.168.1.111:/root_nfs
ip=192.168.1.70:192.168.1.111:192.168.1.111:255.255.255.0:QQ2440.arm9.net:eth0:
off" (NFSの設定)
```

```
Supervivi> boot ram (メモリでカーネルを起動させます)
```

第七章 NOR Flash のブートロードを更新

一般的に NOR Flash のブートロードを更新する必要はないです。

NOR Flash は H-JTAG というツールで更新されます。

H-JTAGはARMの為のJTAGエミュレータです。AXD又はkeilをサポートします。デバッグのスピードも速いです。詳しい情報はこちらです。

<http://www.hitag.com>

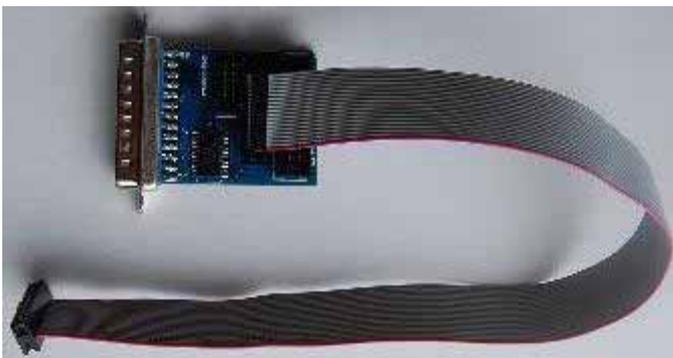
弊社はH-JTAGのハードウェアを提供しております。パソコンはLTPが必要です。

7.1 H-JTAG をダウンロードとインストールします

ホームページ<http://www.hitag.com>から最新版をダウンロードできます。

H-JTAGの特性：

- a. RDI 1.5.0 & 1.5.1 をサポートします；
- b. ARM7 & ARM9 (ARM9E-SとARM9EJ-Sを含む) ；
- c. thumb & arm 命令；
- d. little-endian & big-endian；
- e. semihosting；
- f. 実行環境WINDOWS 9.X/NT/2000/XP；
- g. flashの書き込み

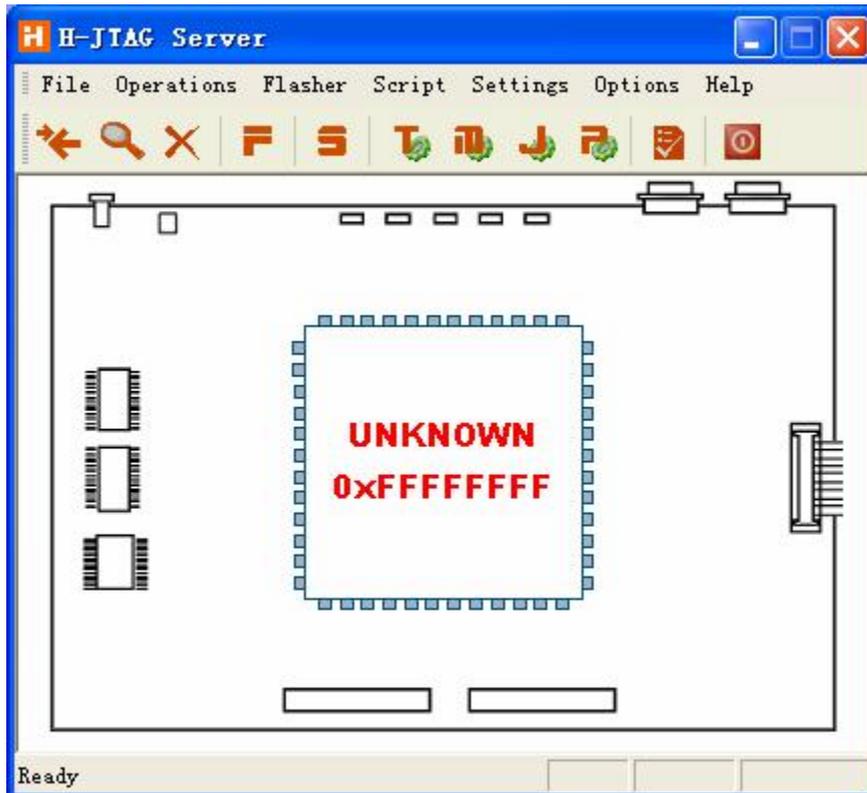


弊社は H-JTAG のハードウェアを提供しております。パソコンは LTP が必要です。

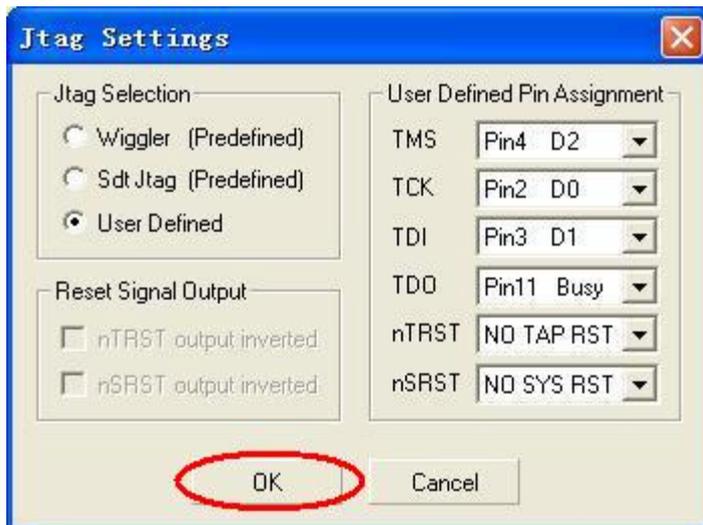
インストール完了すれば、デスクトップで H-JTAG と H-Flasher を生成します。H-JTAG を実行すると、このエラーメッセージが出てきます。



設定しないから。"Ok"ボタンを押すと、初の画面が出てきます。

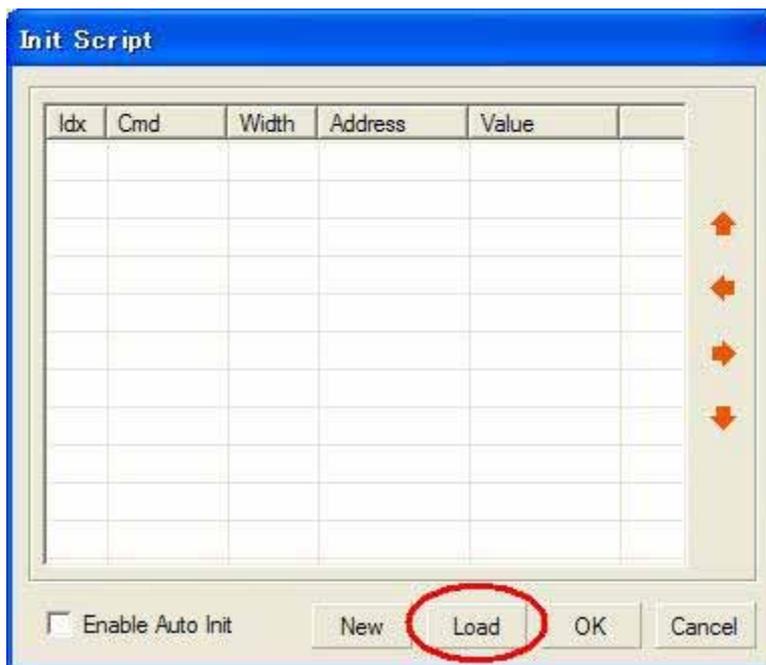


H-JTAG のメニュー : Setting → Jtag Settings

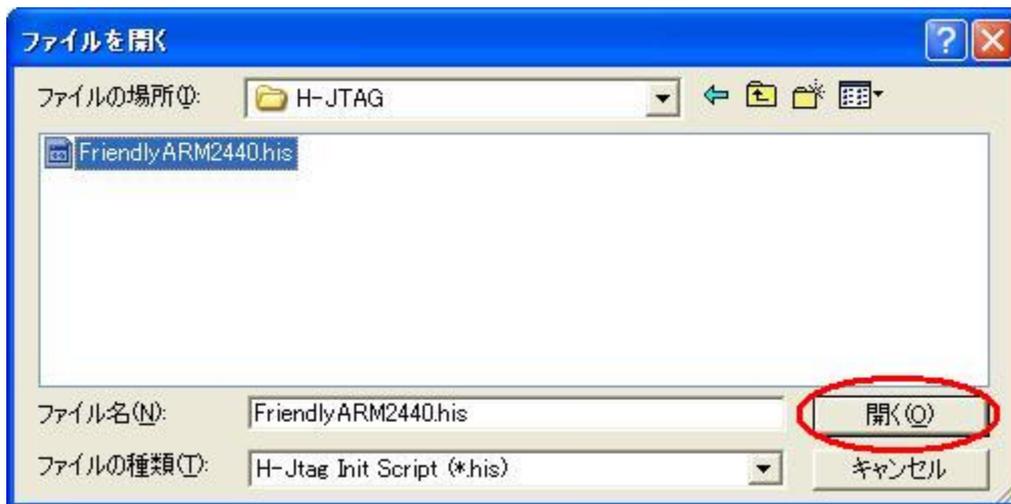


この様な設定して、"Ok"ボタンを押します。

H-JTAG のメニュー : Script → Init Script

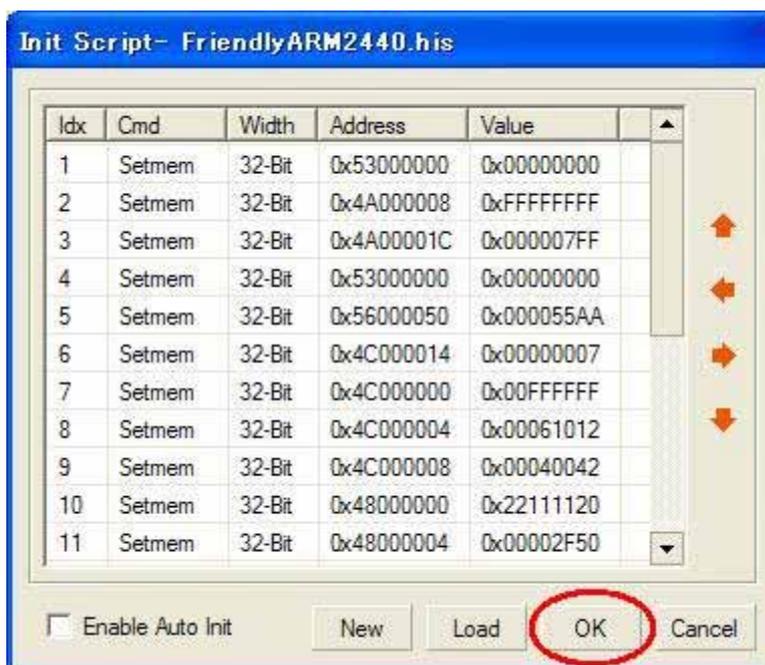


"Load"ボタンを押します。



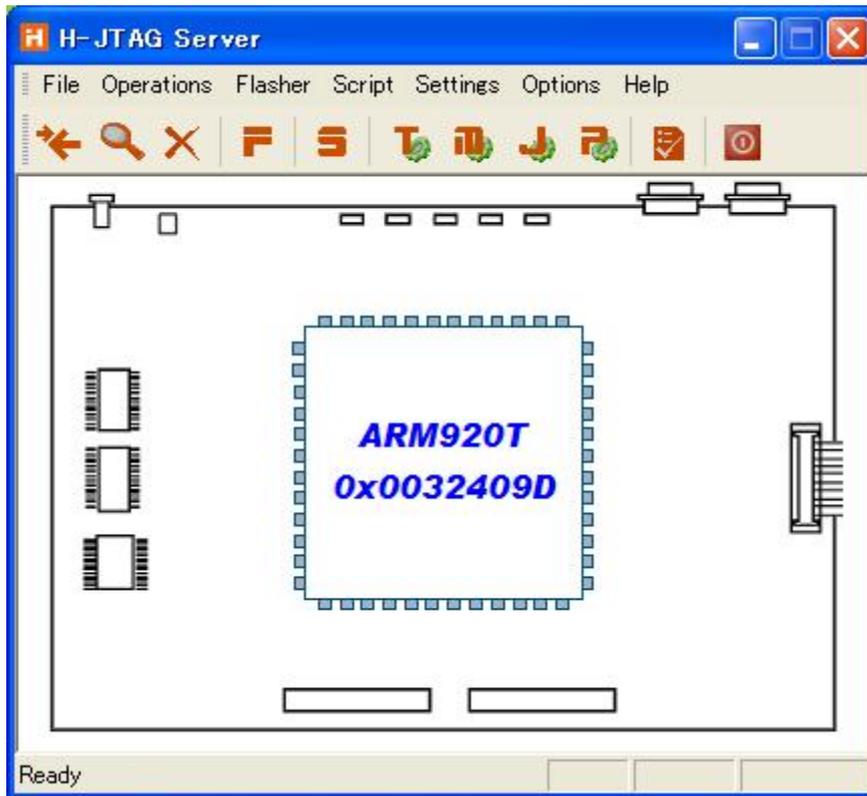
FriendlyARM2440.his というファイルを選択します。

次の画面が出てきます。



"Ok"ボタンを押します。 **ご注意： "Enable Auto Init"をチェックしない。**

パソコンと ARM9 ボードを H-JTAG で繋ぎます。 ARM9 ボードの電源を入れます。
H-JTAG のメニュー： Operations → Detect Target を選択すると

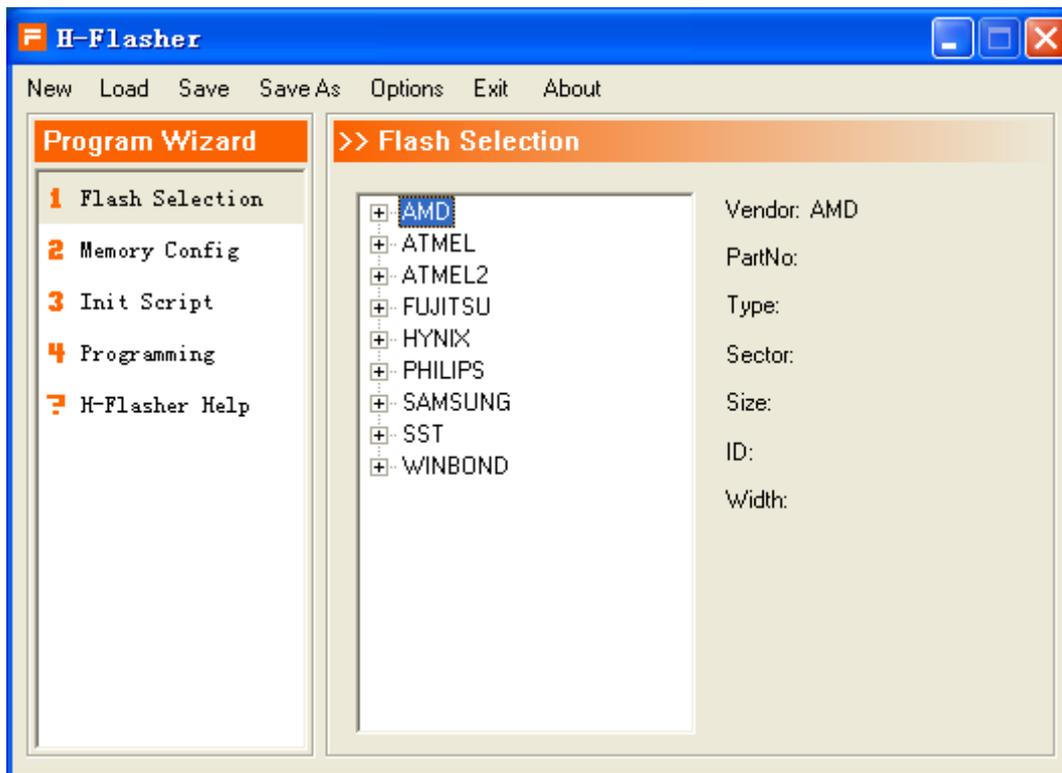


H-JTAG はターゲット ARM ボードを認識しました。

7.2 NOR Flash を書き込む

ARM9 ボードが NOR Flash から起動することを確認してください。

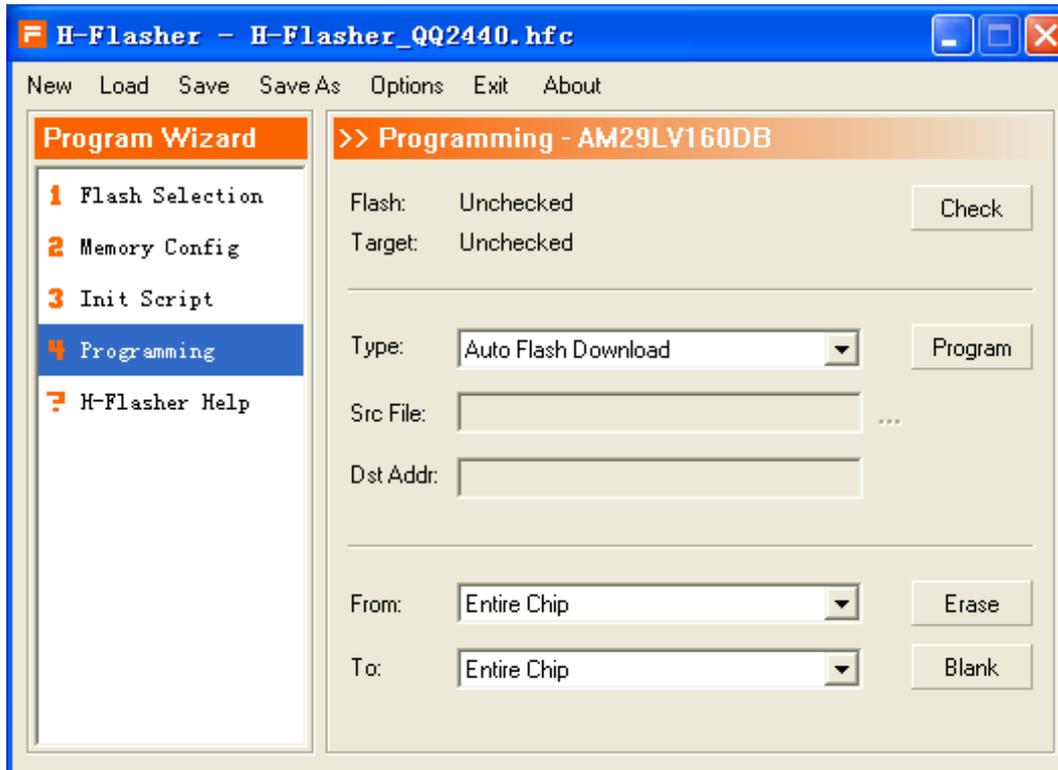
H-JTAG のメインメニュー「Flasher」→「Start H-Flasher」で H-Flasher を実行します。



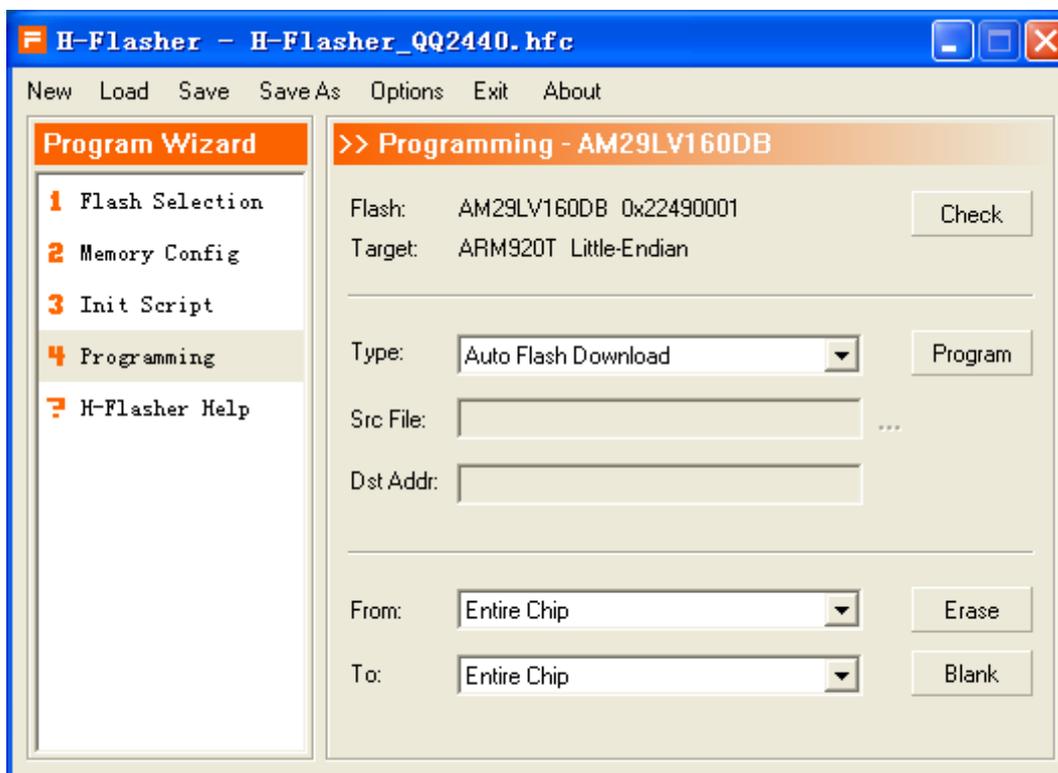
H-Flasherのメインメニュー「Load」、H-Flasher_qq2440.hfcというファイルを開きます。



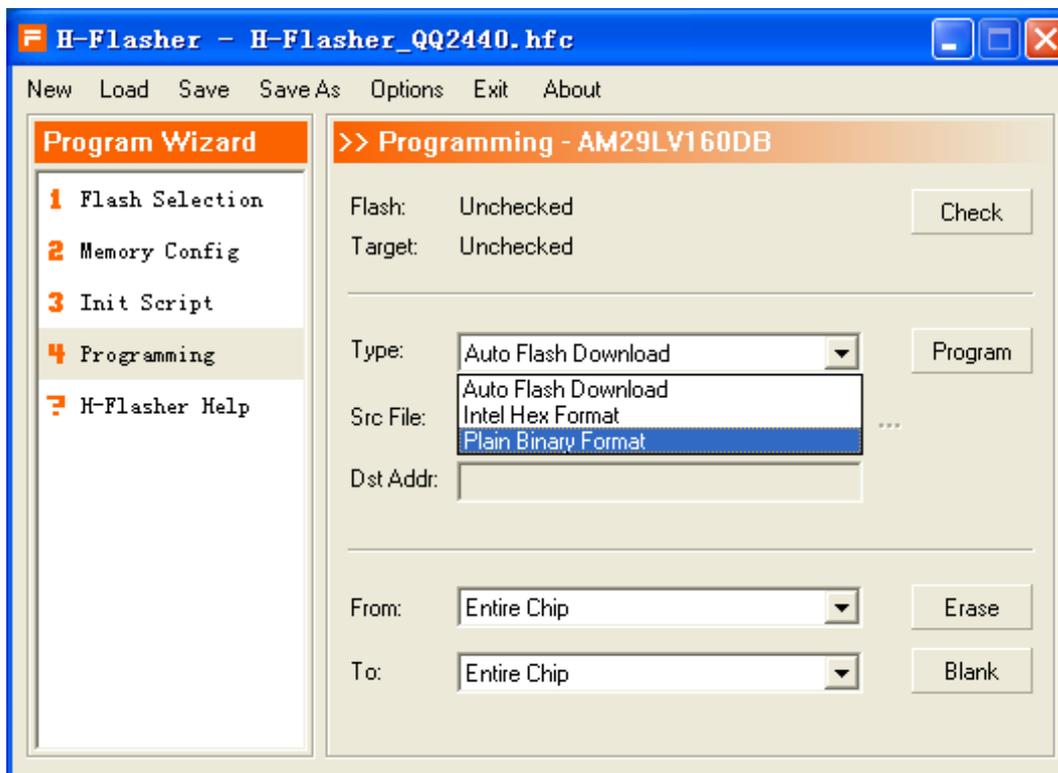
H-Flasherの左側の「4 Programming」を選択します。



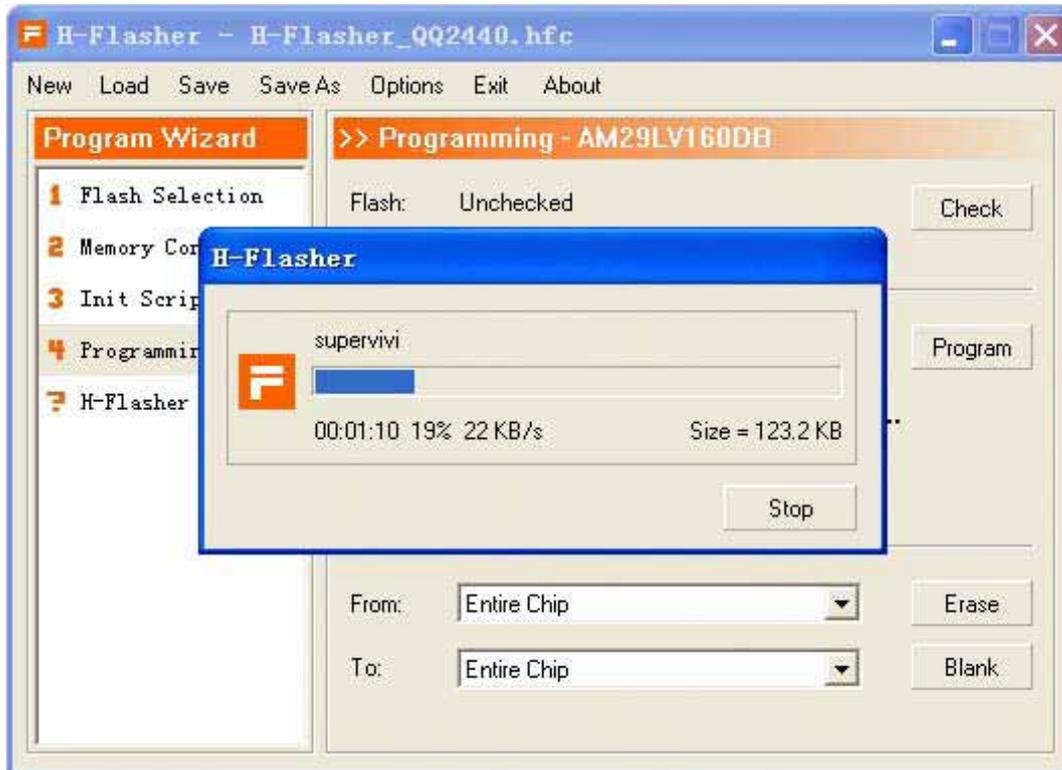
「Check」ボタンを押すと、QQ2440v3が使用したNor Flash(AM29LV160DB)を発見します。



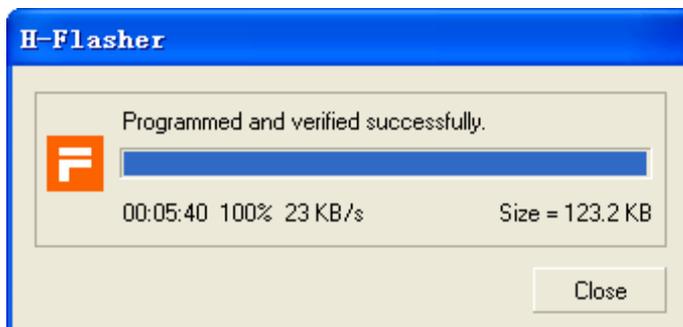
「Type」の「Plain Binary Format」を選択します。



書き込みのファイルsuperviviを選択します。「Dst Addr」で0を入力します。「Program」ボタンを押すと、Nor Flashに書き込みます。



書き込み完了の画面：



付録 1 Qt/Embedded GUI プログラムを作る

ARM9ボードのGUIはQt/Embeddedを使用しています。他のGUIシステムも使えます。例えばminiGUI, ucGUIなど

=====Qt/Embeddedの紹介=====

Qt/Embeddedは、ひとつのソースコード(シングルソース) から複数のプラットフォームで稼働するアプリケーションを開発することができるC++ GUI ツールキットであり、Unix、Linux、Windows、Mac OS X、組み込みLinux 用のGUI アプリケーションを構築するこ



とができます。

その優れた開発コンセプトは、広く世界中のソフトウェア技術者に評価されており、今日では、GPL版、商用版合わせて、世界中に15万人の開発者がいると言われています。また、Qtの採用実績は、大学教育、宇宙開発、特種映像技術、医療、科学技術シュミレーションと多岐にわたり、すでに数千にも及ぶ商用アプリケーションが開発されています。

Qt Embeddedは、組み込みLinuxに特化して提供されるQtで、メモリやCPUなどのマシンリソースが少ない環境でも使用することができます。デスクトップ版のQtは描画の基本的な部分にXサーバーを使用しますが、Xはある程度のマシンリソースを必要としますので、可能な限りハードウェアの実装を小さくしたり、消費電力を抑える必要がある製品には向きません。Qt EmbeddedはLinuxカーネルのAPIを使用し、フレームバッファへ直接描画しますので、Xを必要としません。また、使用する機能だけを使ってプログラム構築することができますので、実行プログラムのフットプリントを最適化することもできます。

詳しい情報はこちらです：

<http://trolltech.com/lang/japanese/qt-embedded>

=====ARM9のQtのサンプル=====

ARM9のソースコードで三つQtパッケージがあります。

x86-qtopia.tgz：X86で実行するパッケージ。

arm-qtopia.tgz：ARMで実行するパッケージ、USBマウスに対応する。

ipaq-qtopia.tgz：ARMで実行するパッケージ、タッチパネルに対応する。

三つのパッケージのソースコードが同じです。buildスクリプトだけが異なります。

1. ソースコードを解凍します：

```
#tar xvzf x86-qtopia.tgz -C /opt/FriendlyARM/mmini2440
```

```
#tar xvzf arm-qtopia.tgz -C /opt/FriendlyARM/mini2440
```

```
#tar xvzf ipaq-qtopia.tgz -C /opt/FriendlyARM/mini2440
```

パソコンで仮想実行するため、/etc/ld.so.confファイルを直します。次の内容に変更します。

```
/opt/FriendlyARM/mini2440/x86-qtopia/qt/lib
```

```
/opt/FriendlyARM/mini2440/x86-qtopia/qtopia/lib
```

```
/usr/kerberos/lib
```

```
/usr/X11R6/lib
```

```
/usr/lib/sane  
/usr/lib/mysql
```

2. X86プラットフォームのQttopiaをコンパイルする

```
#cd /opt/FriendlyARM/mini2440/x86-qttopia  
#./build-all (長い時間がかかります、我慢してください)  
#./konq_to_qttopia (生成されたブラウザをQttopiaに入ります)  
#ldconfig (生成されたqtとqttopiaライブラリを有効します)
```

3. 仮想スクリーン

qvfbはX86プラットフォームの仮想スクリーンです。生成されたQtはこの仮想スクリーンで動きます。

qvfbのインストールの方法：

```
$ tar zxvf qvfb-1.1.tar.gz  
$ cd qvfb-1.1  
$ ./configure  
$ make  
$ su -c 'make install'
```

3. X86プラットフォームで仮想実行する

```
#. set-env ( "."と"set-env"の間、スペースが必要です。 )  
# qvfb &  
# qpe
```



qvfbのメニュー「File」→「Configure」で仮想スクリーンの分解能を設定できます。

4. Hello,Worldをコンパイルする

```
#cd /opt/FriendlyARM/mini2440/x86-qtopia/hello
```

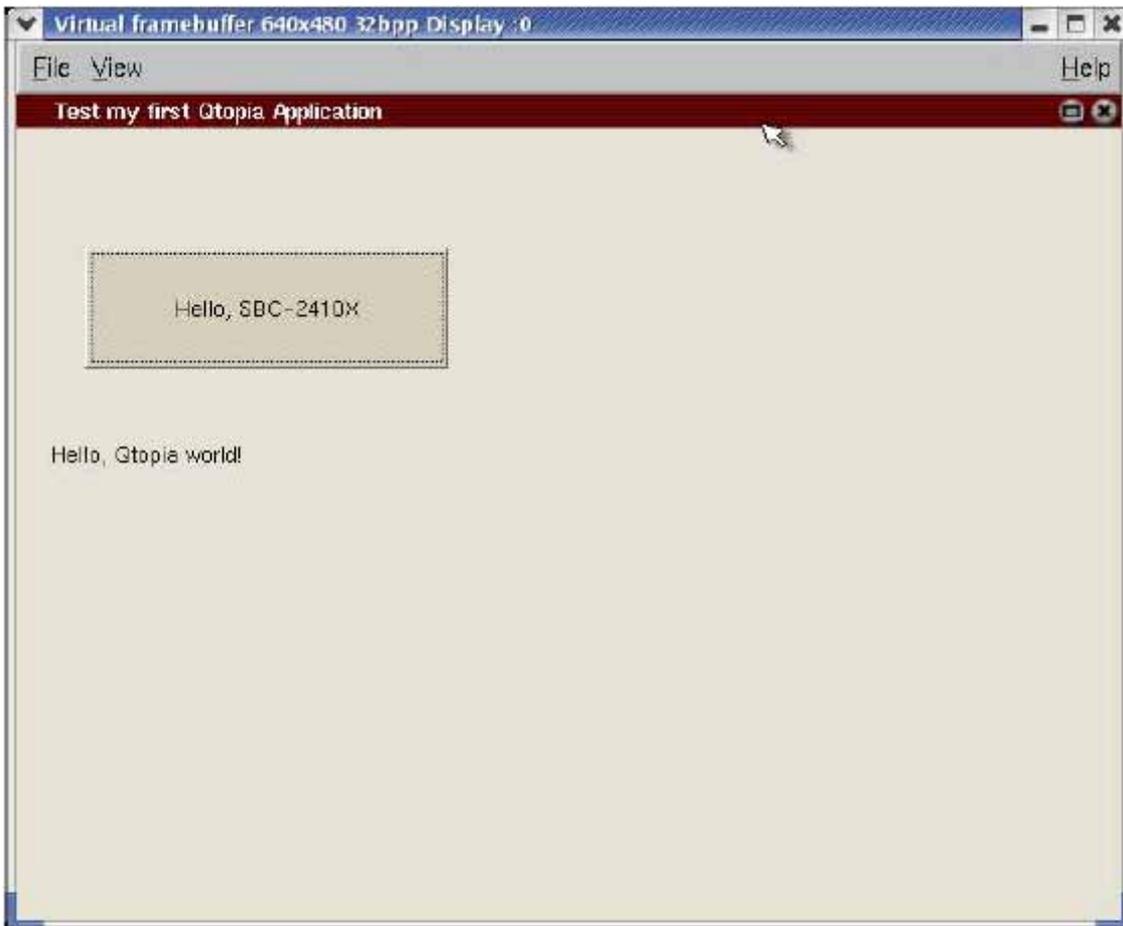
```
#make
```

x86-qtopia/qtopia/binフォルダに実行ファイルhelloを生成します。

5. 単独にHello,Worldを実行する

```
#qxfb -width 640 -height 480 &
```

```
#hello -qws
```



6. Qtopiaの環境でHello,Worldを実行する

```
#cd hello
```

```
#cp hello.desktop /opt/FriendlyARM/mini2440/x86-qtopia/qtopia/apps/Applications
```

```
#qxfb -width 640 -height 480 &
```

```
#qpe
```



7. ARM9プラットフォームのQtopiaをコンパイルする

開発環境はRedhat9をお勧めします。他のLinuxではエラーがあるかもしれません。

クロス開発ツールはarm-linux-gcc-3.3.2です。他のgccはエラーがあるかもしれません。

```
#cd /opt/FriendlyARM/mini2440/arm-qtopia
```

```
#!/build-all (長い時間がかかります、我慢してください)
```

```
#!/mktarget_qtopia (target_qtopia.tgzを生成する)
```

```
#!/mktarget_konq(target_konq.tgzを生成する)
```

8. Hello,Worldをコンパイルする

```
#cd /opt/FriendlyARM/mini2440/arm-qtopia
```

```
#. set-env 'と' set-env'の間に必ずスペースを入ります。
```

```
#cd hello
```

```
#make
```

arm-qtopia/qtopia/binフォルダに実行ファイルhelloを生成します。

9. ARM9でhelloを実行します：

ARM9のコンソール：

```
#mv /mnt/opt/FriendlyARM/qq2440/ipaq-qtopia/qtopia/bin/hello /opt/qtopia/bin
```

```
#mv /mnt/opt/FriendlyARM/qq2440/ipaq-qtopia/hello/hello.desktop
```

```
/opt/qtopia/apps/Applications
```

/mntはARM9とパソコンの共有フォルダです。

ARM9を再起動すれば、helloのアイコンが見えます。実行すると：



10. 単独にARM9でhelloを実行します：

ARM9のコンソール：

```
export set HOME=/root
```

```
export set QTDIR=/opt/qt
```

```
export set QPEDIR=/opt/qtopia
```

```
export set QWS_KEYBOARD="USB:/dev/input/event1"
```

```
export set QWS_MOUSE_PROTO="USB:/dev/input/mouse0"
```

```
export set PATH=$QPEDIR/bin:$PATH
```

```
export set LD_LIBRARY_PATH=$QTDIR/lib:$QPEDIR/lib
```

```
$QPEDIR/bin/hello -qws
```

11. 自分で生成されたQtopiaをファイルシステムに組み込む

```
#cd /opt/FriendlyARM/mini2440/root_qtopia_mouse
```

```
#tar xvzf ../arm-qtobia/target_qtobia.tgz (Qtobiaを更新する)
#tar xvzf ../arm-qtobia/target_konq.tgz (ブラウザを更新する)
#mkyaffsimage root_qtobia_mouse my_qtobia.img
生成されたmy_qtobia.imgファイルをARM9ボードに書き込みます。
```

付録2 アプリケーションを移植

Linux環境にたくさんのオープンソースのアプリケーションがありますが、これらは殆んどパソコン(x86)用のものです。どうやってパソコン用のものをARM9に移植しますか。ある例を通じて、紹介します。

ARM9ボードの起動の時、自動的にあるMP3をディスプレイします。このツールはmadplayです。madplayのウェブサイトはこちらです。

<http://www.underbit.com/products/mad/>

このウェブサイトの紹介によって、madplayはライブラリlibmadとlibid3tagが必要です。ウェブサイトからこの三つのソースファイルをダウンロードします。

madplay-0.15.2b.tar.gz

libmad-0.15.1b.tar.gz

libid3tag-0.15.1b.tar.gz

多分他のソースファイルも必要するかもしれません。移植中に問題があれば、だんだん添加します。

まず、パソコンでコンパイルしましょう。

step1: ソースファイルをsrc-x86に解凍します

```
mkdir src-x86
```

```
mkdir target-x86
```

```
tar xvzf libid3tag-0.15.1b.tar.gz -C src-x86
```

```
tar xvzf libmad-0.15.1b.tar.gz -C src-x86
```

```
tar xvzf madplay-0.15.2b.tar.gz -C src-x86
```

step2: ライブラリをコンパイルします

```
cd ../src-x86/libid3tag-0.15.1b
```

```
./configure --prefix=/madplay-source/target-x86
```



```
make  
make install
```

```
cd ../libmad-0.15.1b  
./configure --prefix=/madplay-source/target-x86  
make  
make install
```

コンパイル成功したら、target-x86でincludeとlibを生成します。

step3: madplayをコンフィグします

```
cd madplay-0.15.2b  
./configure --prefix=/madplay-source/target-x86  
そうすると、エラーが出てきます。  
configure: error: mad.h was not found  
*** You must first install libmad before you can build this package.  
*** If libmad is already installed, you may need to use the CPPFLAGS  
*** environment variable to specify its installed location, e.g. -I<dir>.
```

エラー情報によって、madplayをコンフィグする前にlibmadをインストールする必要があります。step2でlibmadのincludeを生成しましたから、CPPFLAGS環境変数を設定してみます。

```
./configure --prefix=/madplay-source/target-x86  
CPPFLAGS=-I/madplay-source/target-x86/include
```

もうエラーが出てきます。

```
configure: error: libmad was not found  
*** You must first install libmad before you can build this package.  
*** If libmad is already installed, you may need to use the LDFLAGS  
*** environment variable to specify its installed location, e.g. -L<dir>.
```

エラー情報によって、ライブラリの環境変数LDFLAGSの設定も必要です。もう一度コンフィグしてみます。

```
./configure --prefix=/madplay-source/target-x86  
CPPFLAGS=-I/madplay-source/target-x86/include  
LDFLAGS=-L/madplay-source/target-x86/lib
```



今回コンフィグ成功しました。

step4: madplayをコンパイルします

コンフィグ成功したら、コンパイルできます。

```
make
```

```
make install
```

成功したら、target-x86/binで実行ファイルmadplayが見えます。

step5: madplayをテストします

あるMP3ファイルをmadplayディレクトリにコピーします。

```
./madplay test.mp3
```

このコマンドすると、MP3が聞こえます。

これまで、madplayをパソコンに移植完了しました。今までの操作ステップとコマンドも覚えますか。忘れないように、これらコマンドをあるスクリプトに書きます。

```
#!/bin/sh
```

```
MADPLAY_DIR=$PWD
```

```
SRC_DIR=src-x86
```

```
TARGET_DIR=$MADPLAY_DIR/target-x86
```

```
tar xvzf ./tarball/libid3tag-0.15.1b.tar.gz -C $SRC_DIR
```

```
tar xvzf ./tarball/libmad-0.15.1b.tar.gz -C $SRC_DIR
```

```
tar xvzf ./tarball/madplay-0.15.2b.tar.gz -C $SRC_DIR
```

```
cd $SRC_DIR/libid3tag-0.15.1b
```

```
./configure --prefix=$TARGET_DIR
```

```
make;make install
```

```
cd ../
```

```
cd $SRC_DIR/libmad-0.15.1b
```

```
./configure --prefix=$TARGET_DIR
```

```
make;make install
```



```
cd ../../
```

```
cd $SRC_DIR/madplay-0.15.2b
./configure --prefix=$TARGET_DIR CPPFLAGS=-I$TARGET_DIR/include
LDLFLAGS=-L$TARGET_DIR/lib
make;make install
cd ../../
```

このスクリプトに基づいて、ARM9に移植してみよう。

ARMの移植なら、ARMのコンパイルツールを指定する必要があります。デフォルトコンパイルツールはgccです。更新されたスクリプト(赤い文字はARMの為の更新)：

```
#!/bin/sh
```

```
MADPLAY_DIR=$PWD
SRC_DIR=src-arm
TARGET_DIR=$MADPLAY_DIR/target-arm
```

```
tar xvzf ./tarball/libid3tag-0.15.1b.tar.gz -C $SRC_DIR
tar xvzf ./tarball/libmad-0.15.1b.tar.gz -C $SRC_DIR
tar xvzf ./tarball/madplay-0.15.2b.tar.gz -C $SRC_DIR
```

```
export CC=arm-linux-gcc
```

```
cd $SRC_DIR/libid3tag-0.15.1b
./configure --host=arm-linux --prefix=$TARGET_DIR
CPPFLAGS=-I$TARGET_DIR/include LDLFLAGS=-L$TARGET_DIR/lib
make;make install
cd ../../
```

```
cd $SRC_DIR/libmad-0.15.1b
./configure --host=arm-linux --prefix=$TARGET_DIR
make;make install
cd ../../
```

```
cd $SRC_DIR/madplay-0.15.2b
```



```
./configure --host=arm-linux --prefix=$TARGET_DIR  
CPPFLAGS=-I$TARGET_DIR/include LDFLAGS=-L$TARGET_DIR/lib  
make;make install  
cd ../../
```

このスクリプトを実行すると、エラーも出てきます。原因はlibid3tagをコンパイル失敗しました。libid3tagのコンパイルはライブラリ zlibに依存します。x86の環境でzlibがあるので、別途必要ないです。ARMの環境にはありません。ウェブサイトでzlibをダウンロードしてみよう。もう一度スクリプトを更新します(青い文字はzlibに関連します)。

```
#!/bin/sh
```

```
MADPLAY_DIR=$PWD  
SRC_DIR=src-arm  
TARGET_DIR=$MADPLAY_DIR/target-arm
```

```
tar xvzf ./tarball/libid3tag-0.15.1b.tar.gz -C $SRC_DIR  
tar xvzf ./tarball/libmad-0.15.1b.tar.gz -C $SRC_DIR  
tar xvzf ./tarball/madplay-0.15.2b.tar.gz -C $SRC_DIR  
tar xvzf ./tarball/zlib-1.2.3.tar.gz -C $SRC_DIR
```

```
export CC=arm-linux-gcc
```

```
cd $SRC_DIR/zlib-1.2.3  
./configure --prefix=$TARGET_DIR  
make && make install  
cd ../../
```

```
cd $SRC_DIR/libid3tag-0.15.1b  
./configure --host=arm-linux --prefix=$TARGET_DIR  
CPPFLAGS=-I$TARGET_DIR/include LDFLAGS=-L$TARGET_DIR/lib  
make;make install  
cd ../../
```

```
cd $SRC_DIR/libmad-0.15.1b  
./configure --host=arm-linux --prefix=$TARGET_DIR
```

```
make;make install  
cd ../../
```

```
cd $SRC_DIR/madplay-0.15.2b  
./configure --host=arm-linux --prefix=$TARGET_DIR  
CPPFLAGS=-I$TARGET_DIR/include LDFLAGS=-L$TARGET_DIR/lib  
make;make install  
cd ../../
```

今回ARM環境の実行ファイルを生成了しました。target-arm/binで実行ファイルmadplayが見えます。生成されたmadplayとライブラリをFTPでARM9ボードの/usr/bin/と/usr/lib/にダウンロードします。

ARM9ボードでmadplayをテストしてみよう！

付録3 Watchdog の使い方

ウォッチドッグ タイマ (Watchdog Timer) は組み込みシステムの大切な機能です。システムフリーズ状態になった時点で再起動 (reboot) を実行させることができます。OSなし環境でウォッチドッグ タイマをよく使いますが、Linux環境でどうやって使用しますか。

step1:

Linuxのコンフィグ

kernel-2.6.13でmake menuconfig

メニュー-Device Drivers → Character devices → Watchdog Cards → Watchdog Timer Support → S3C2410 Watchdogを"M"に設定します。

***'M' の設定はLinuxカーネルを更新しなくても、ドライバをダイナミックロードすることができます。'*'に設定すると、ドライバはLinuxカーネルの一部としてコンパイルします。LinuxカーネルをARM9ボードのFlashに再び書き込みしなければなりません。**

step2:

kernel-2.6.13でmake modulesコマンドでドライバをコンパイルします。

成功すれば、オブジェクトファイルkernel-2.6.13/drivers/char/watchdog/s3c2410_wdt.koを生成します。

step3:

ARM9ボードでドライバをダイナミックロードします。

```
insmod s3c2410_wdt.ko
```

```
mknod /dev/watchdog c 10 130
```

ls /dev/コマンドでwatchdogが見えるはずですが。

step4:

ARM9ボードでwatchdogを起動します。

```
echo "@" > /dev/watchdog
```

長い間echo "@" > /dev/watchdogをしないと、watchdogが動作して、ARM9ボードを再起動させます。

詳しいWatchdogの使い方、こちらに参照してください。

Linux カーネルのドキュメント:

kernel-2.6.13/Documentation/watchdog/watchdog.txt

付録 4 AD の使い方

ソース : readadc.c

実行ファイル : readadc

ADCのドライバ

ソース : sbc2440_adc_sample.c

実行ファイル : sbc2440_adc_sample.ko

ドライバのコンパイル :

sbc2440_adc_sample.cをlinux/drivers/char/にコピーしてください。

linux/drivers/char/Kconfigファイルに次の内容を入力してください。

```
config SBC2440_ADC
```

```
    tristate "SBC2440 ADCs Driver"
```

```
    depends on ARCH_S3C2410
```

```
    help
```

SBC2440 ADC.

linux/drivers/char/Makefileファイルに次の内容を入力してください。

```
obj-$(CONFIG_SBC2440_ADC) += sbc2440_adc_sample.o
```

make menuconfigでADCのドライバが見えます。[M]を選択した、makeで再コンパイルすれば、linux/drivers/char/にsbc2440_adc_sample.koを生成させます。

ARM9ボードでADCドライバのインストール：

```
# insmod sbc2440_adc_sample.ko  
# mknod /dev/sbc2440_adc_sample0 c 236 0
```

ADCドライバのテスト

```
# ./readadc
```

付録 5 UVC(USB Device Class)Web カメラを使用

USB ビデオクラスとは

読み方：ユーエスビービデオクラス

別名：USB Video Device Class

【英】USB video class

USB ビデオクラスとは、USB 2.0 の規格が拡張されたもので、家庭用ビデオカメラ、またはライブカメラなどに用いられる Web カメラなどの、映像を扱う機器を USB でパソコンに接続するための規格のことである。

USB 規格を策定する組織である USB Implementers Forum によって策定された。Motion-JPEG、DV、MPEG-2 ないしは MPEG-4 などの圧縮データを、あるいは非圧縮のデータを、USB を通じて転送することが可能となる。

UVC カメラを ARM9 で使用できるソフト

```
# tar zxvf arm-linux-uvc.tgz
```

arm-linux-uvc の中に次のファイルがあります。

uvccideo-r104.tar.gz: uvccideo ドライバのソースファイル。

mjpg_streamer_rev35.tgz: ストリーミングの配信用のソースファイル。

libjpeg62: MJPG-streamer をコンパイルするためのライブラリと.h ファイル

arm9-bin: クロスコンパイルされる実行ファイルです。

uvccideo.ko: uvccideo ドライバ

*.so: MJPG-streamer のライブラリ

mjpg_streamer: MJPG-streamer の実行ファイル

www: MJPG-streamer のホームページ

ソフトを ARM9 ボードにインストールする手順

これらのファイルを ARM9 ボードに FTP でアップロードします。

UVC(USB Device Class)に対応した Web カメラを ARM9 の USB ポートに接続して、ARM9 ボードの環境で次のコマンドを入力します。

まず、特定のディレクトリにファイルをコピーします。

```
# cp arm9-bin/*.so /usr/lib/
```

```
# cp arm9-bin/mjpg-streamer /usr/bin/
```

次に、uvccideo.ko をカーネルに組み込みます。

```
# insmod arm9-bin/uvccideo.ko
```

```
uvccideo: Found UVC 1.00 device USB 2.0 Camera (0c45:62c0)
```

```
usbcore: registered new driver uvccideo
```

```
USB Video Class driver (v0.1.0)
```

最後に、以下のコマンドで MJPG-streamer を起動します。

```
# mjpg_streamer --input "input_uvc.so --device /dev/video0 --fps 5 --resolution  
640x480 --yuv" --output "output_http.so --port 8080 --www  
/mjpg-streamer/www"
```

Web ブラウザで ARM9 ボードを見ましょう

Web ブラウザで、「http://ARM9 の IP アドレス):8080/」にアクセスすると、MJPG-Streamer Demo Pages が表示されます。静止画、動画、および Pan/Tilt/LED の On/Off 等の制御をすることができます。(Internet Explorer 6 及び 7 では、MJPEG によるストリーム(動画)を閲覧することができません。しかし、Javascript を使用したストリーム(動画)は、Internet Explorer でも閲覧することができます。)

Web ブラウザで見る様子:



付録 6 Mini GUI の移植とプログラム設計

1. MiniGUI 概要

MiniGUI は、複数のオペレーティング・システム対応の、組み込みデバイス向けグラフィカル・ユーザ・インタフェース開発システムおよびミドルウェアとして代表的なものです。MiniGUI は、アジアの代表的なテレコムやエンターテインメント機器メーカーの多くに採用されており、携帯デバイス(携帯電話、ポータブル・メディア・プレイヤー、PDA)、セットトップ・ボックス、医療機器、産業用制御システムおよび軍事システムにも幅広く



利用されています。

MiniGUI は、高速で安定性がある軽量 GUI フレームワークおよび開発システムで、リソースやリアルタイム・パフォーマンスに厳しい制約のあるアプリケーション向けに最適化されています。MiniGUI は成熟したマルチウィンドウイング/メッセージング・メカニズムを備え、LCD や YUV など様々なデバイスと連携し、拡張 GDI API (すなわち、ラスター、複雑な描画/作図、2D グラフィックス) をサポートし、フラッシュ GUI 構築用のスキン API を提供します。

MiniGUI は、static labels, ボタン, edit boxes(単一またはマルチライン), list and combo boxes, progress bars, property sheets, toolbars, track bars, tree, list, grid, icon views、月例カレンダー、アニメーションといった、各種コントロール機能(ウィジェット)の包括的なスイートを提供します。MiniGUI は、dialog boxes, message boxes, menus, acceleration keys, carets, timers を備えています。また、広く普及している画像ファイル・タイプ(すなわち、GIF、JPEG、BMP、PNG)、Windows リソースファイル(すなわち、Windows ビットマップ、アイコン、カーソル)、複数の文字セット、フォント・タイプ、キーボード・レイアウトをサポートします。

詳しい情報はこちらです。

<http://www.minigui.org>

2. X86 プラットフォームで実行してみよう

MiniGUI は他のライブラリを使用していますので、コンパイルする前に、PNG/JPEG/TTF などライブラリをインストールする必要があります。

2.1 PNG ライブラリをコンパイルとインストール

```
$ tar zxvf libpng_src.tgz
$ cd libpng
$ cp scripts/makefile.linux Makefile
$ make
$ su -c 'make install'
```

2.2 JPEG ライブラリをコンパイルとインストール

```
$ tar zxvf jpegsrc.v6b.tar.gz
$ cd jpeg-6b
```

```
$ ./configure  
$ make  
$ su -c 'make install'
```

2.3 TrueTypeFont ライブラリをコンパイルとインストール

```
$ tar zxvf freetype-1.3.1.tar.gz  
$ mkdir -p libttf/extend  
$ cp freetype-1.3.1/lib/* freetype-1.3.1/lib/arch/ansi/* libttf/  
$ cp freetype-1.3.1/lib/extend/* libttf/extend/  
$ cp freetype-1.3.1/lib/* freetype-1.3.1/lib/arch/ansi/* libttf/  
$ cd libttf  
  
$ gcc -c -fPIC -O2 freetype.c  
$ gcc -c -fPIC -O2 -I./ extend/*.c  
$ gcc --shared -o libttf.so *.o  
  
$ su -c 'cp libttf.so /usr/local/arm/2.95.3/arm-linux/lib'
```

2.4 MiniGUI をコンパイルとインストール

```
$ tar zxvf libminigui-1.6.10.tar.gz  
$ cd libminigui-1.6.10  
$ ./configure  
$ make  
$ su -c 'make install'
```

2.5 MiniGUI のソースをインストール

```
$ tar zxvf minigui-res-1.6.10.tar.gz  
$ cd minigui-res-1.6.10  
$ su -c 'make install'
```

/etc/ld.so.conf ファイルを編集して、MiniGUI のインストール先/usr/local/lib を入ります。ルートアカウントで次のコマンドでシステムのライブラリを更新します。

```
# ldconfig
```

2.6 仮想スクリーン

MiniGUI は X86 プラットフォームで実行すれば、仮想スクリーンが必要です。

```
$ tar zxvf qvfb-1.1.tar.gz
```

```
$ cd qvfb-1.1
```

```
$ ./configure
```

```
$ make
```

```
$ su -c 'make install'
```

```
$ qvfb &
```

qvfb のメニュー 「File」 → 「Configure」 で仮想スクリーンの分解能を設定できます。

2.7 MiniGUI のデモ

MiniGUI はデモ、ゲーム、サンプルがあります。

デモ : mde-1.6.10.tar.gz

ゲーム : games-1.6.10.tar.gz

```
$ tar zxvf mde-1.6.10.tar.gz
```

```
$ cd mde-1.6.10
```

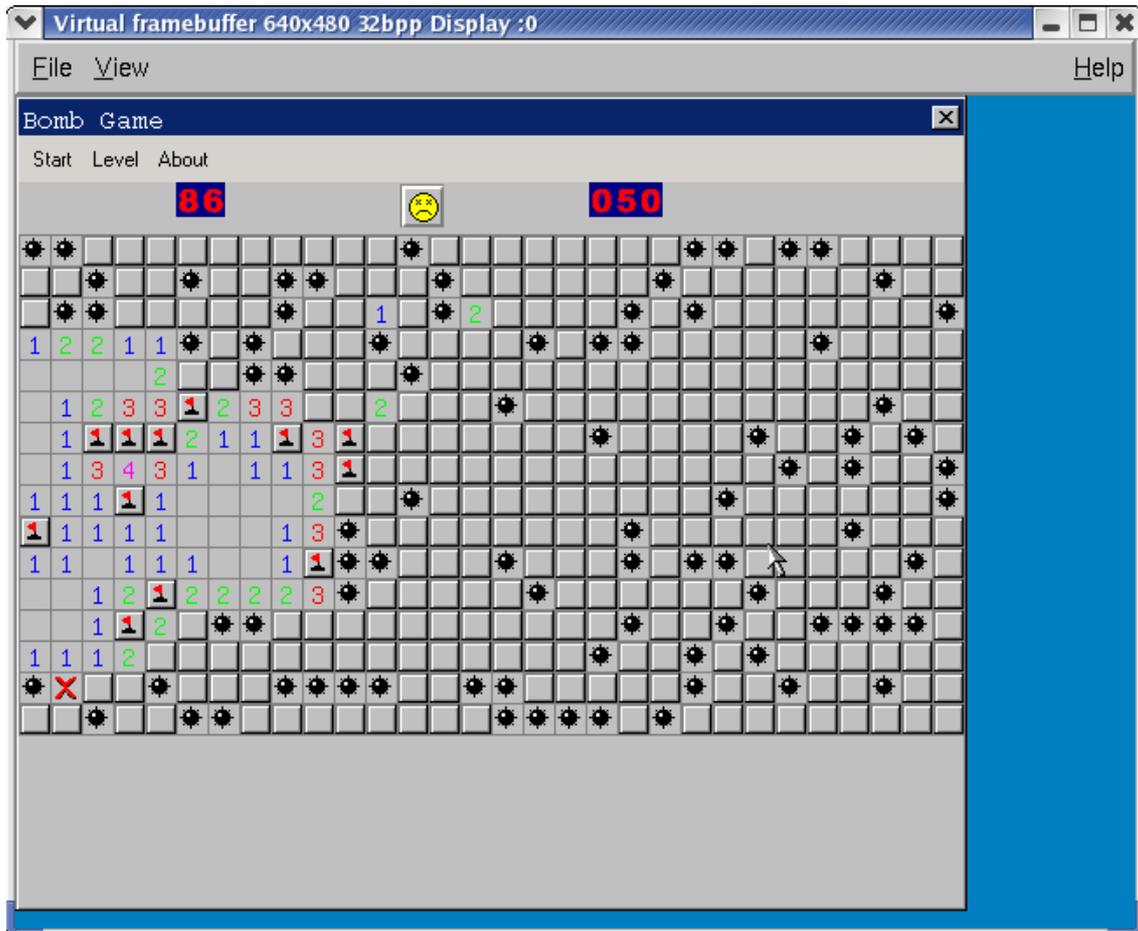
```
$ ./configure
```

```
$ make
```

フォルダの中で幾つの例を生成します。一つの例を実行してみよう。

```
$ cd bomb
```

```
$ ./bomb
```



Windows の画面みたいです。他の例をやってみましょう。

2.8 MiniGUI 環境のプログラム設計

MiniGUIのウェブサイトでは<http://www.minigui.org/docs.shtml>

参考資料を提供しています。ご覧ください。

サンプル : mg-samples-1.6.10.tar.gz

3. ARM9 プラットフォームに移植する

インストール手順は X86 と大体同じです。ARM9 で幾つのライブラリがないので、追加することが必要です。MiniGUI のクロス開発ツールは 2.95.3 を使用します。

3.1 Z ライブラリをコンパイルとインストール

Z ライブラリは他のライブラリの基礎です。X86 のデフォルト設定は Z ライブラリがあり

ますが、ARM9 がありません。まず、Z ライブラリをインストールします。

```
$ tar zxvf zlib-1.2.3.tar.gz
$ cd zlib-1.2.3
$ export CC=arm-linux-gcc      #コンパイルツールを arm に変換する
$ ./configure --prefix=/usr/local/arm/2.95.3/arm-linux/ --shared
$ make
$ su -c 'make install'
```

3.2 PNG ライブラリをコンパイルとインストール

```
$ tar zxvf libpng_src.tgz
$ cd libpng
$ cp scripts/makefile.linux Makefile
```

Makefile ファイルを編集して、次の内容に変更してください

```
CC=arm-linux-gcc
prefix=/usr/local/arm/2.95.3/arm-linux
ZLIBLIB = /usr/local/arm/2.95.3/arm-linux/lib
ZLIBINC = /usr/local/arm/2.95.3/arm-linux/include
```

```
$ make
$ su -c 'make install'
```

3.3 JPEG ライブラリをインストール

JPEG のソースパッケージの configure のなかにバグがあるので、コンパイルするのは大変です。そして、直接に arm-linux-gcc-3.3.2 の JPEG ライブラリをコピーします。

```
su -c 'cp /usr/local/arm/3.3.2/arm-linux/lib/libjpeg*.* /usr/local/arm/2.95.3/arm-linux/lib'
```

3.4 TrueTypeFont ライブラリをコンパイルとインストール

```
tar zxvf freetype-1.3.1.tar.gz
mkdir -p libttf/extend
cp freetype-1.3.1/lib/* freetype-1.3.1/lib/arch/ansi/* libttf/
cp freetype-1.3.1/lib/extend/* libttf/extend/
cp freetype-1.3.1/lib/* freetype-1.3.1/lib/arch/ansi/* libttf/
cd libttf
arm-linux-gcc -c -fPIC -O2 freetype.c
arm-linux-gcc -c -fPIC -O2 -I./ extend/*.c
```

```
arm-linux-gcc --shared -o libttf.so *.o
su -c 'cp libttf.so /usr/local/arm/2.95.3/arm-linux/lib'
```

3.5 MiniGUI をコンパイルとインストール

```
$ tar zxvf libminigui-1.6.10.tar.gz
$ cd libminigui-1.6.10
```

configure ファイルを編集して、次の内容を先頭に追加してください

```
CC=/usr/local/arm/2.95.3/bin/arm-linux-gcc
CPP=/usr/local/arm/2.95.3/bin/cpp
LD=/usr/local/arm/2.95.3/bin/arm-linux-ld
AR=/usr/local/arm/2.95.3/bin/arm-linux-ar
RANLIB=/usr/local/arm/2.95.3/bin/arm-linux-ranlib
STRIP=/usr/local/arm/2.95.3/bin/arm-linux-strip
```

```
$ ./configure --host=arm-linux --enable-jpgsupport=yes --enable-pngsupport=yes
--enable-gifsupport=yes --disable-lite --prefix=/usr/local/arm/2.95.3/arm-linux
$ make
$ su -c 'make install'
```

3.6 MiniGUI のソースをインストール

```
$ tar zxvf minigui-res-1.6.10.tar.gz
$ cd minigui-res-1.6.10
```

config.linux ファイルを編集して、次のような変更します。

```
TOPDIR=/minigui/miniguitmp
```

```
$ su -c 'make install'
```

3.7 popt ライブラリをコンパイルとインストール

popt は MiniGUI デモ用のライブラリです。

```
$ tar zxvf popt-1.7.tar.gz
$ cd popt-1.7
$ ./configure --prefix=/usr/local/arm/2.95.3/arm-linux/ --host=arm-linux --enable-shared
--enable-static
$ make
```

```
$ su -c 'make install'
```

3.8 生成されたファイル ARM9 のフォルダにコピー

仮定 ROOT_NFS は ARM9 の NFS ルートファイルシステムです。

次のファイルを /usr/local/arm/2.95.3/arm-linux/lib から ROOT_NFS/usr/lib にコピーしてください。

libjpeg.la	libminigui.so	libpopt.so.0
libjpeg.so	libm.so	libpopt.so.0.0.0
libjpeg.so.62	libm.so.6	libttf.so
libjpeg.so.62.0.0	libpng.so	libz.so
libm-2.2.3.so	libpng.so.2	libz.so.1
libminigui-1.6.so.10	libpng.so.2.1.0.12	libz.so.1.2.3
libminigui-1.6.so.10.0.0	libpopt.la	
libminigui.la	libpopt.so	

MiniGUI 用ソースをコピーします。

```
$ cp --archive /minigui/miniguitmp/usr ROOT_NFS/
```

MiniGUI 用コンフィグファイル MiniGUI.cfg をコピーします。

```
$ cp /usr/local/arm/2.95.3/arm-linux/etc/MiniGUI.cfg ROOT_NFS/usr/local/etc
```

MiniGUI.cfg を編集して、次のような変更します。

```
[system]
# GAL engine and default options
gal_engine=fbcon

# IAL engine
ial_engine=dummy          #デモ用から、入力デバイスを指定しません
mdev=none
mtype=none

[fbcon]
defaultmode=240x320-16bpp  #3.5 インチの LCD、7 インチなら 800x480
```

MiniGUI が動く環境は MiniGUI.cfg に設定されます。MiniGUI はあるファイルを見つからないと、MiniGUI.cfg を直してください。

3.9 デモのインストール

```
$ tar zxvf mde-1.6.10.tar.gz
$ cd mde-1.6.10
$ ./configure --prefix=${ROOT_NFS}/usr/local --host=arm-linux
$ make

$ mkdir -p ${ROOT_NFS}/usr/local/lib/shared/miniguiapps
$ cp -r -a mde-1.6.10/* ${ROOT_NFS}/usr/local/lib/shared/miniguiapps
```

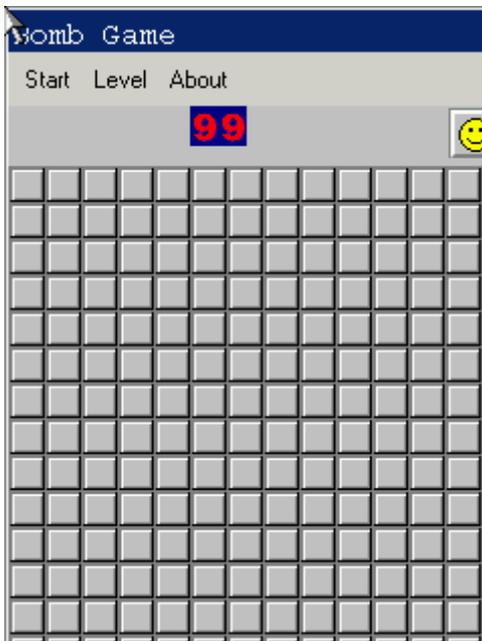
実は実行ファイルと res ファイルだけをコピーするのは十分です。

3.10 ARM9 で実行してみよう

ARM9 ボードは NFS で起動します。ホストの ROOT_NFS は ARM9 のルートファイルシステムです。

ARM9 のコンソール :

```
# cd /usr/local/lib/shared/miniguiapps
# cd bomb
# ./bomb
```



ARM9 ボードで同じ画面が出てきます。

若し次の故障があれば

GAL fbcon engine: Can't open /dev/tty0: No such file or directory

GAL: Init GAL engine failure: fbcon.

GDI: Can not initialize graphics engine!

次のコマンドを実行して、再びデモを実行します。

```
# mknod /dev/tty0 c 4 0
```

4. MiniGUI の IAL(Input Abstract Layer)

MiniGUI は異なる入力デバイス、例えばマウス、キーボード、ボタンなどに対応するため、IAL(Input Abstract Layer)があります。IAL はこれら入力信号を内部統一のメッセージに変換しますので、MiniGUI のアプリケーションはプラットフォームを関連しません。異なるハードウェアでも、異なる OS でも動きます。

自分の IAL プログラムを作る手順：

仮定新 IAL プログラムの名前は `_NAME_IAL` です。IAL のフォルダは `libminigui/src/ial` です。

(1) `ial.c` ファイルに次の内容を添加します。

```
#ifdef _NAME_IAL
    #include "NAME.h"
#endif
```

```
//input の配列の中に
#ifdef _NAME_IAL
    {"NAME ", InitNAMEInput, TermNAMEInput},
#endif
```

(2) 新 IAL プログラムを `Makefile.am` に入れます。

(3) MiniGUI をコンフィグする時、使用したい IAL を指定します。例：

```
$ ./configure --host=arm-linux --enable-jpgsupport=yes --enable-pngsupport=yes
--enable-gifsupport=yes --disable-lite --prefix=/usr/local/arm/2.95.3/arm-linux
--enable-_NAME_IAL=yes
```

(4) IAL プログラムの例

`smdk2410.tar` は mini2440 用 IAL です。3.5 インチタッチパネルと 6 個ボタンの IAL です。参考してください。詳しい作り方は MiniGUI のドキュメントを参照してください。

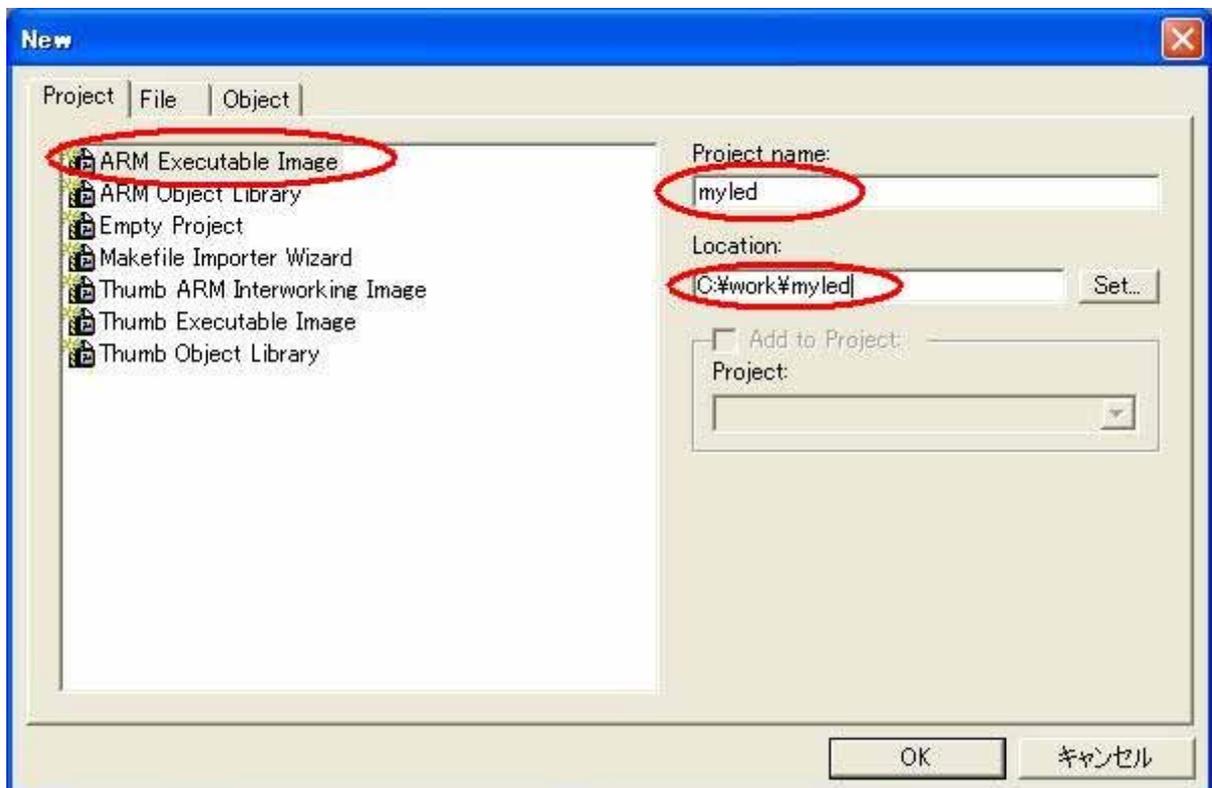
付録7 ADS1.2 と H-JTAG でプログラムを開発

ADS1.2 (ARM Developer Suite) は ARM 社によって開発された ARM 開発ツールです。ADS1.2 で開発されたプログラムは直接に ARM9 にロードできるし、実行できる。ADS1.2 で OS なし環境でプログラムを開発するのは便利です。

ADS1.2 でプログラムを開発することを紹介いたします。

1. 新プロジェクトを作る

ADS のメニュー - File → New

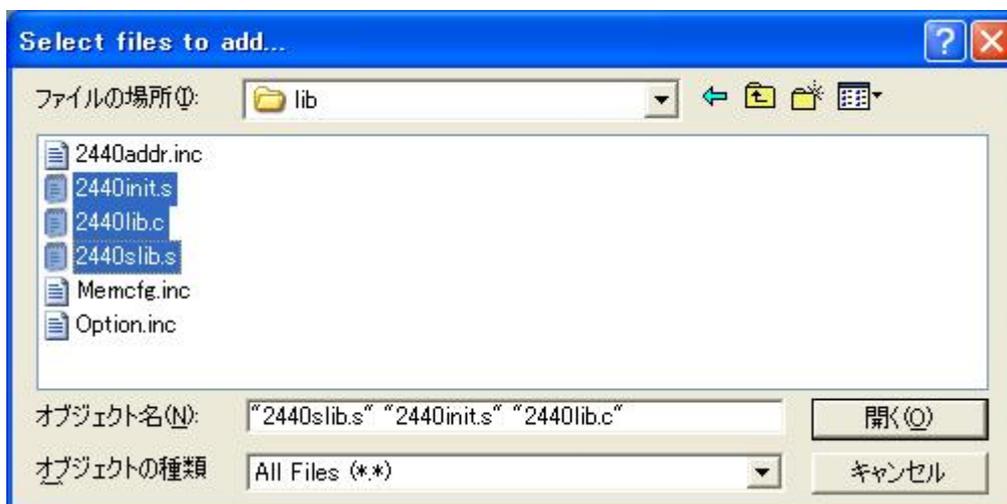


"Project name"と"Location"を入力してください。入力完了すれば、"Ok"ボタンを押して、次のウィンドが出てきます。

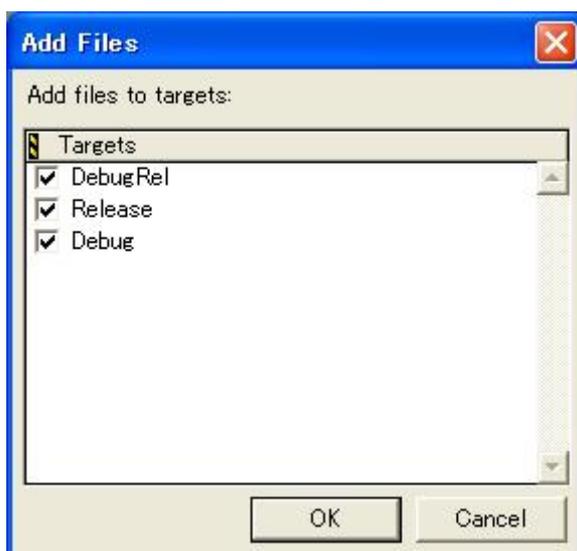


マウスの右ボタンをクリックして、"Add Files"を選択してください。

ソースフォルダ myled/lib 中のファイルを選択してください。



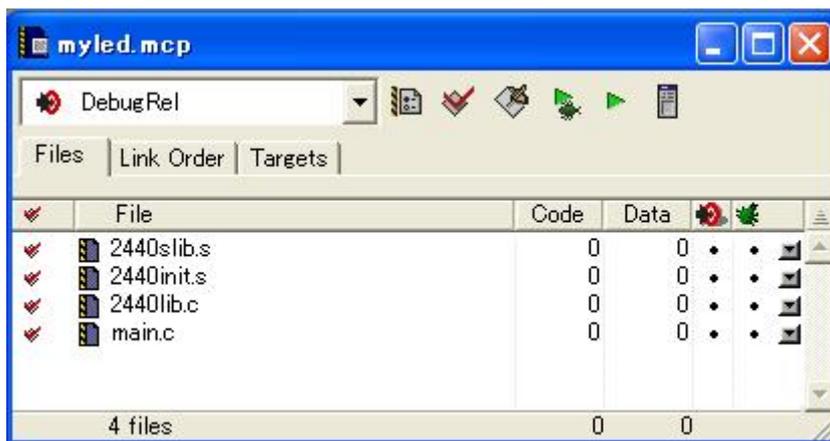
"開く"ボタンを押すと、次のウィンドが出てきます。"Ok"ボタンを押してください。



ソースフォルダ myled 中の main.c も選択してください。

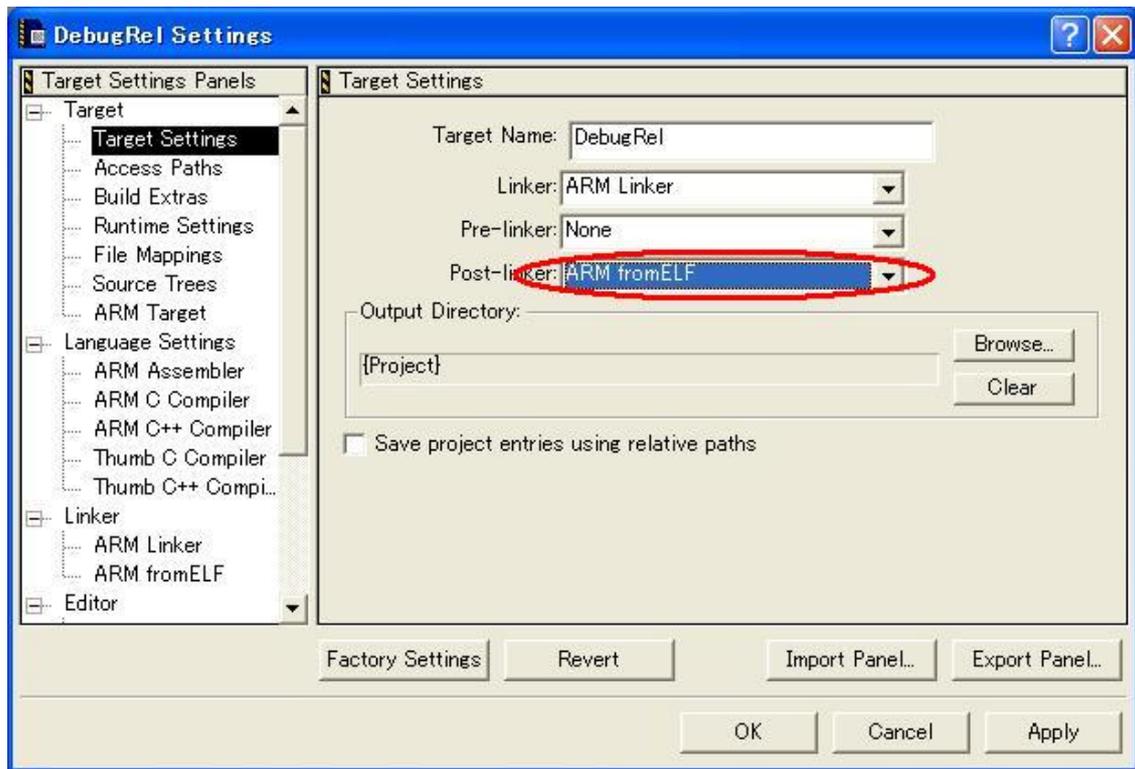


プロジェクトで全てファイルを添加した様子 :

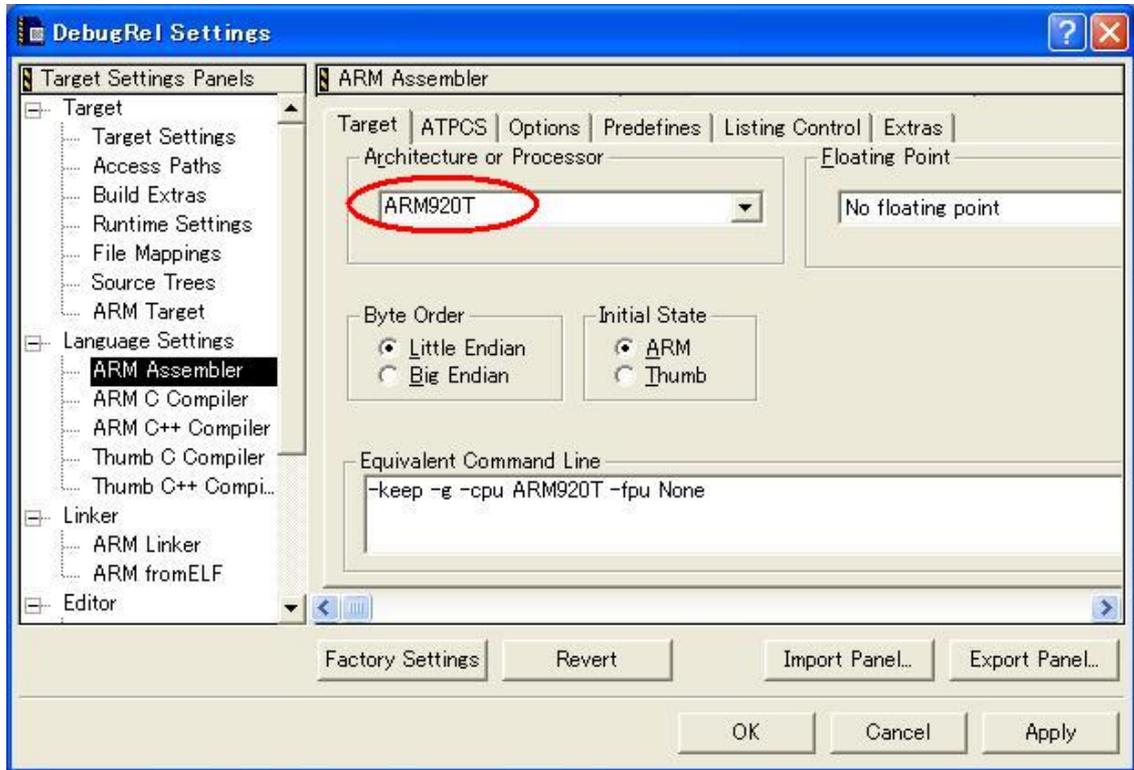


2. コンパイルの環境の設定

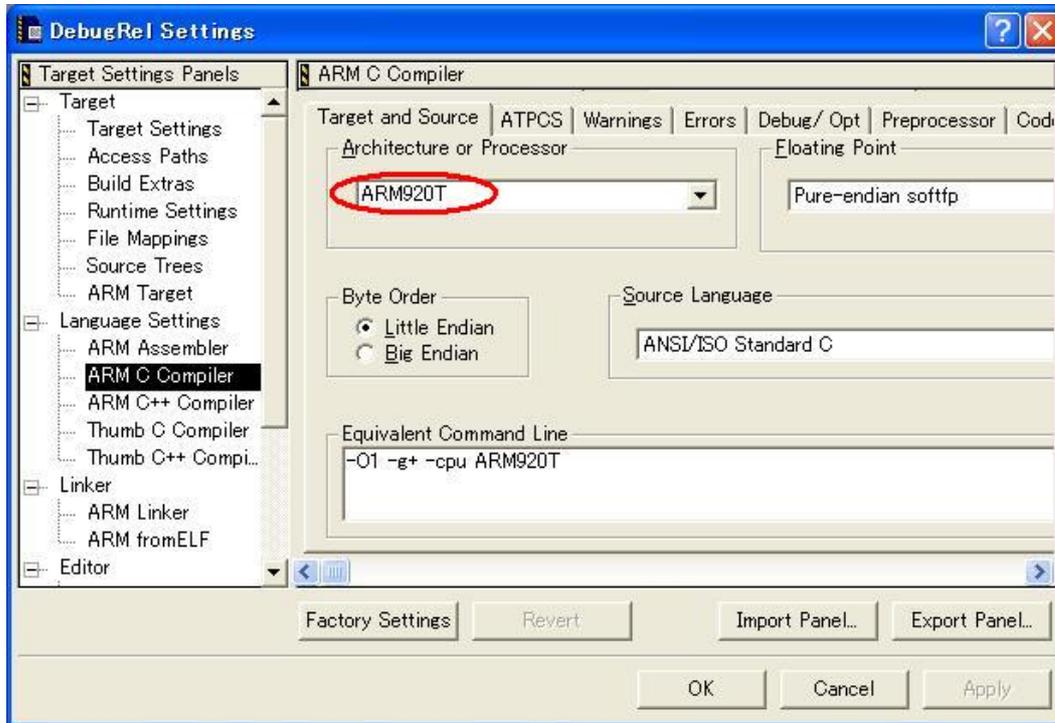
コンパイルの環境の設定：メニューEdit → DebugRel Setting...を選択してください。左側の「Target Settings」を選択して、右側の「Post linker」で「ARM fromELF」を選択します。



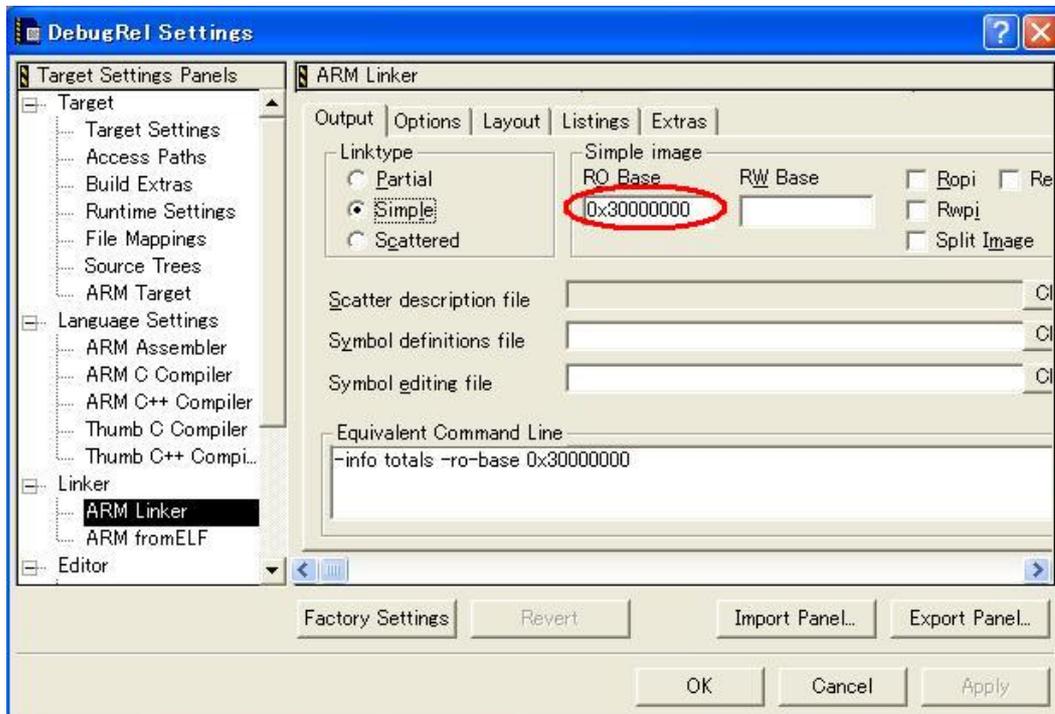
左側の「ARM Assembler」を選択して、アセンブラのコンパイラーを ARM920T に設定してください。



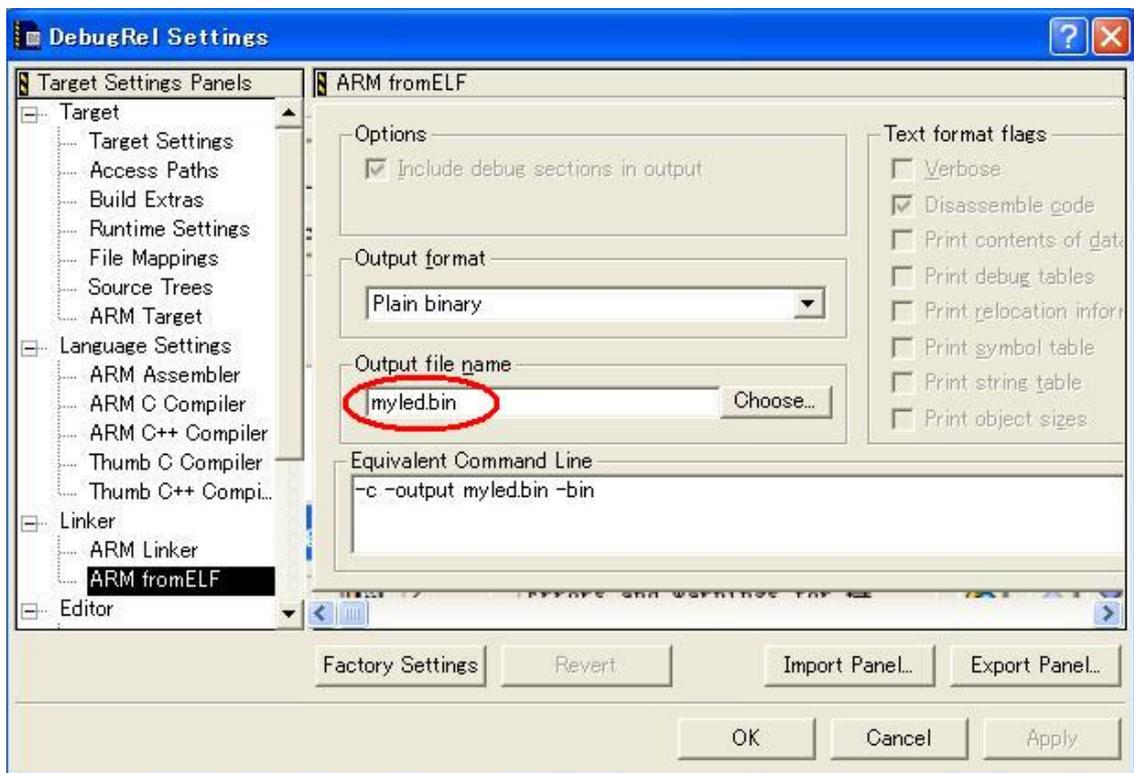
左側の「ARM C Compiler」を選択して、C コンパイラーを ARM920T に設定してください。



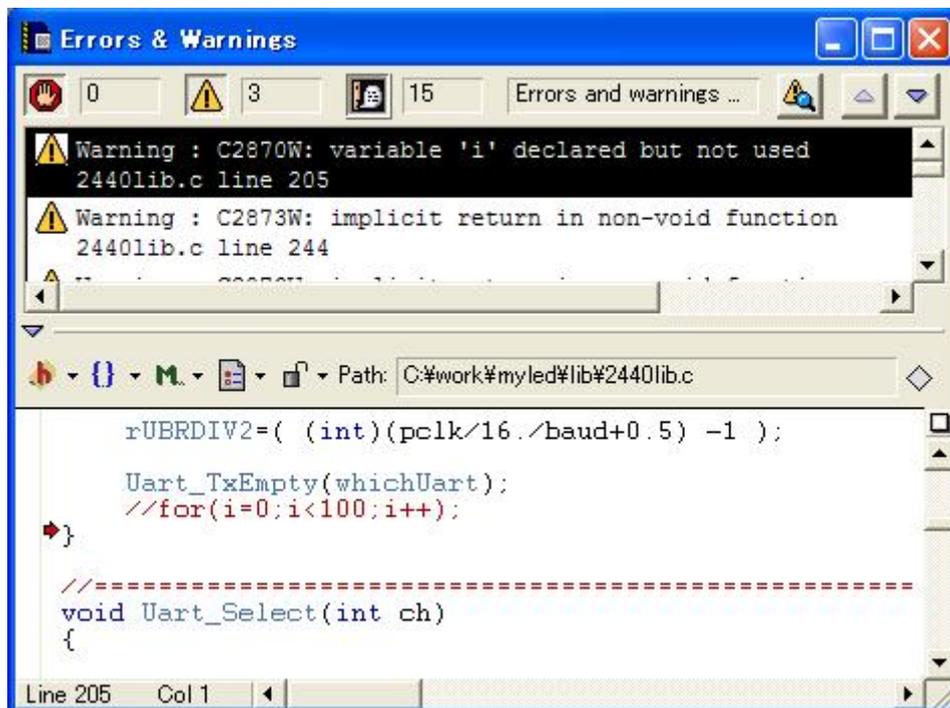
左側の「ARM Linker」を選択して、プログラムの実行アドレスを設定します。ARM9 ボードの場合は 0x30000000 です。



左側の「ARM fromELF」を選択して、出力ファイルの名前を入力してください。



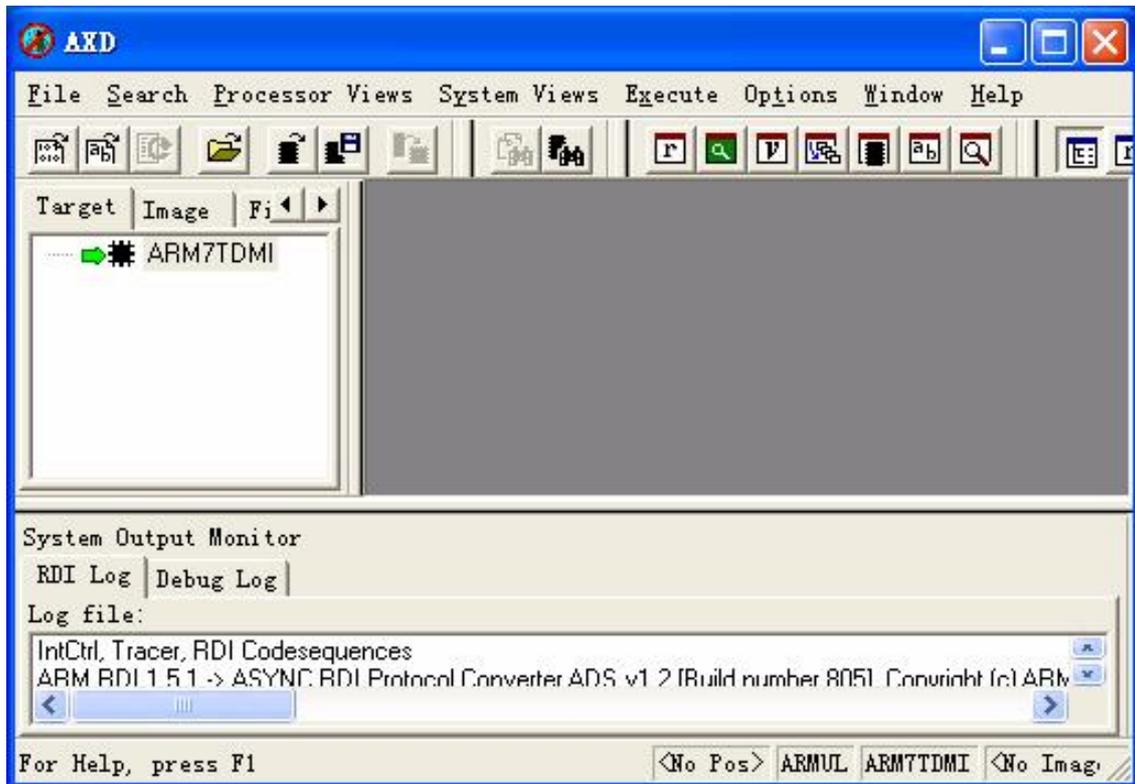
ADSメニュー：Project → make を選択して、実行できる bin ファイルを生成します。



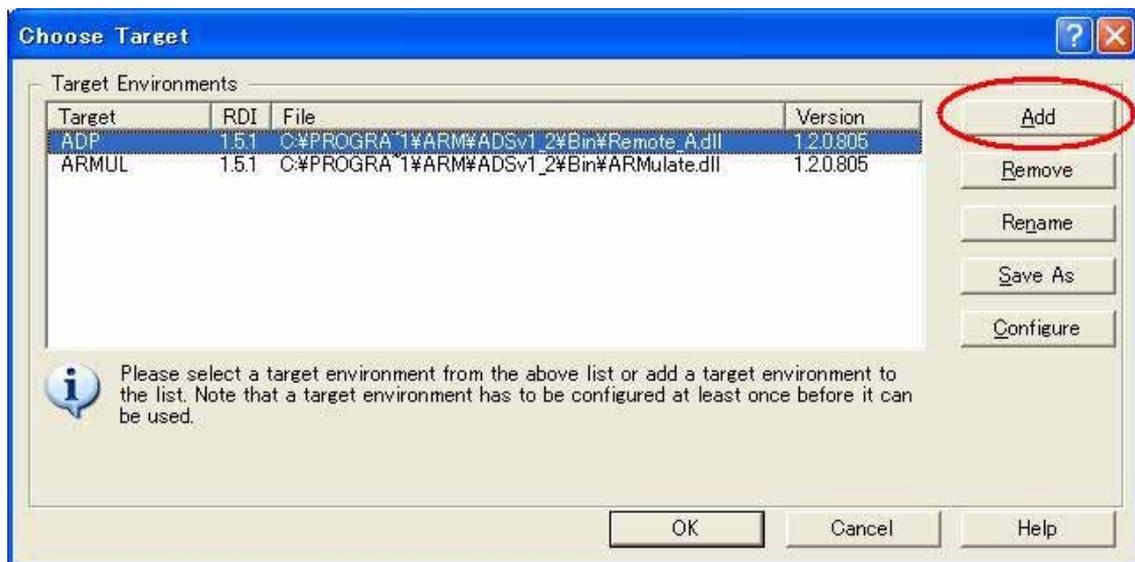
エラーがなければ、myled/myled_Data/DebugRel で myled.bin を生成します。

3. デバッグ環境の設定

ADS1.2 の AXD Debugger を実行します。



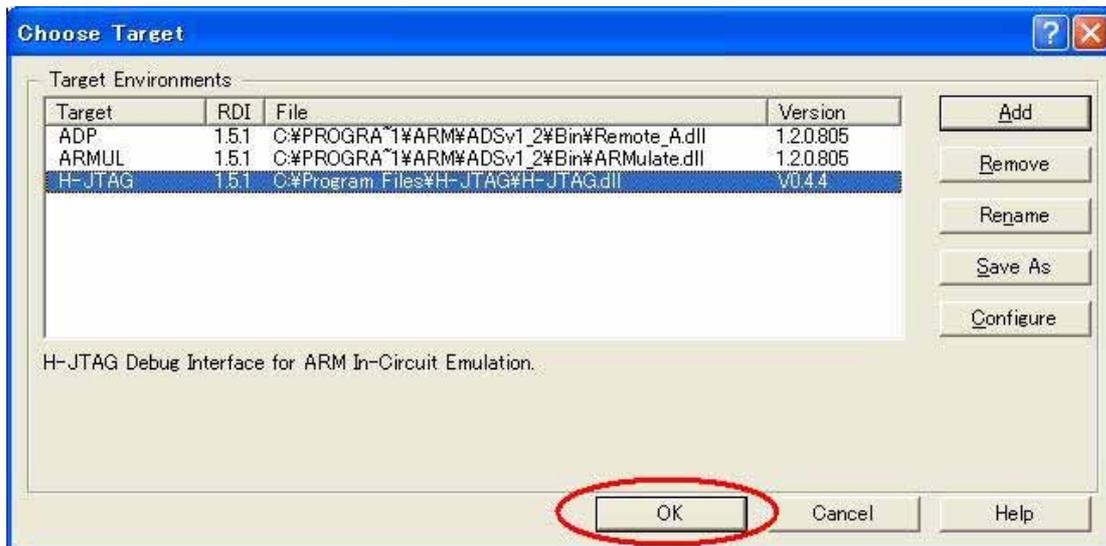
AXD Debugger のメニュー : Options → Configurer Target...



"Add"ボタンを押して、H-JTAG をインストールしたフォルダで H-JTAG.dll ファイルを添加します。



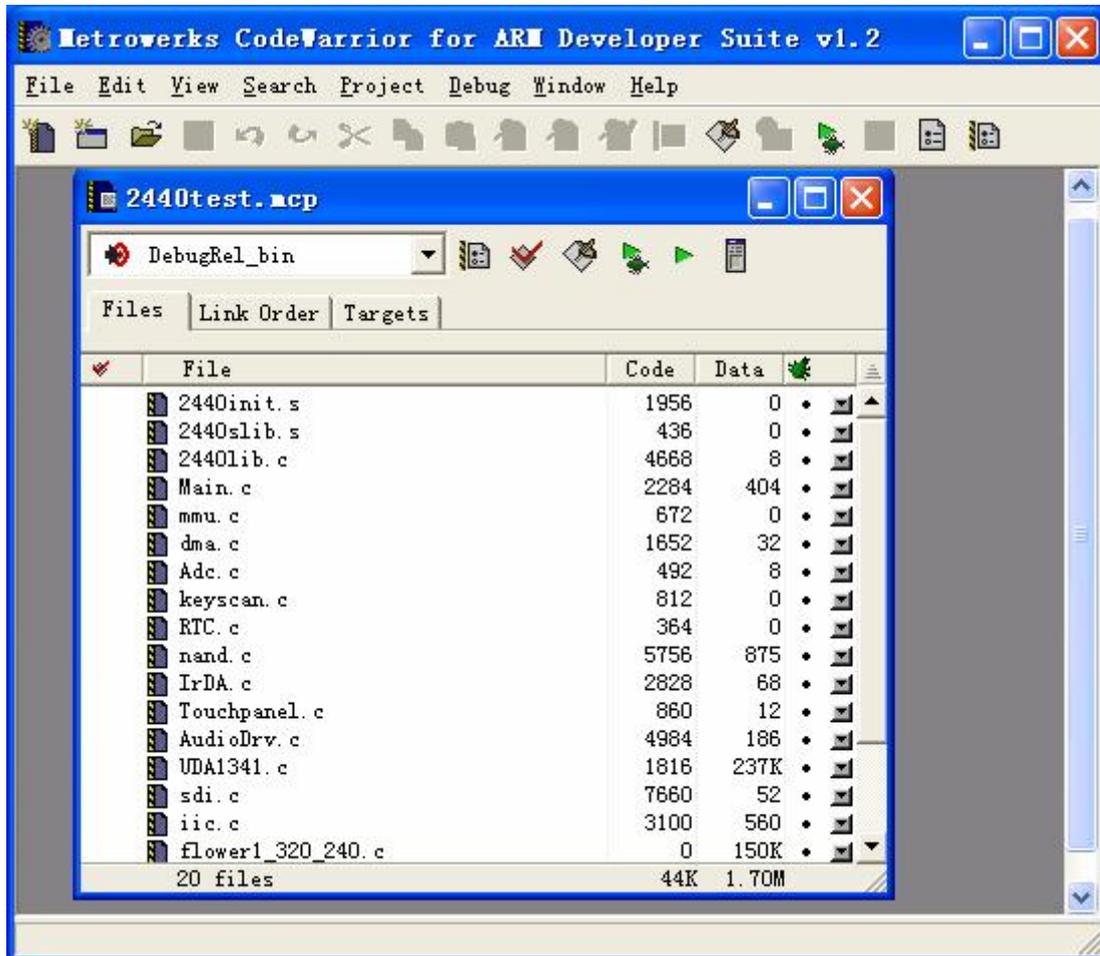
H-JTAG.dll を添加した様子。



4. H-JTAG でプログラムをデバッグします

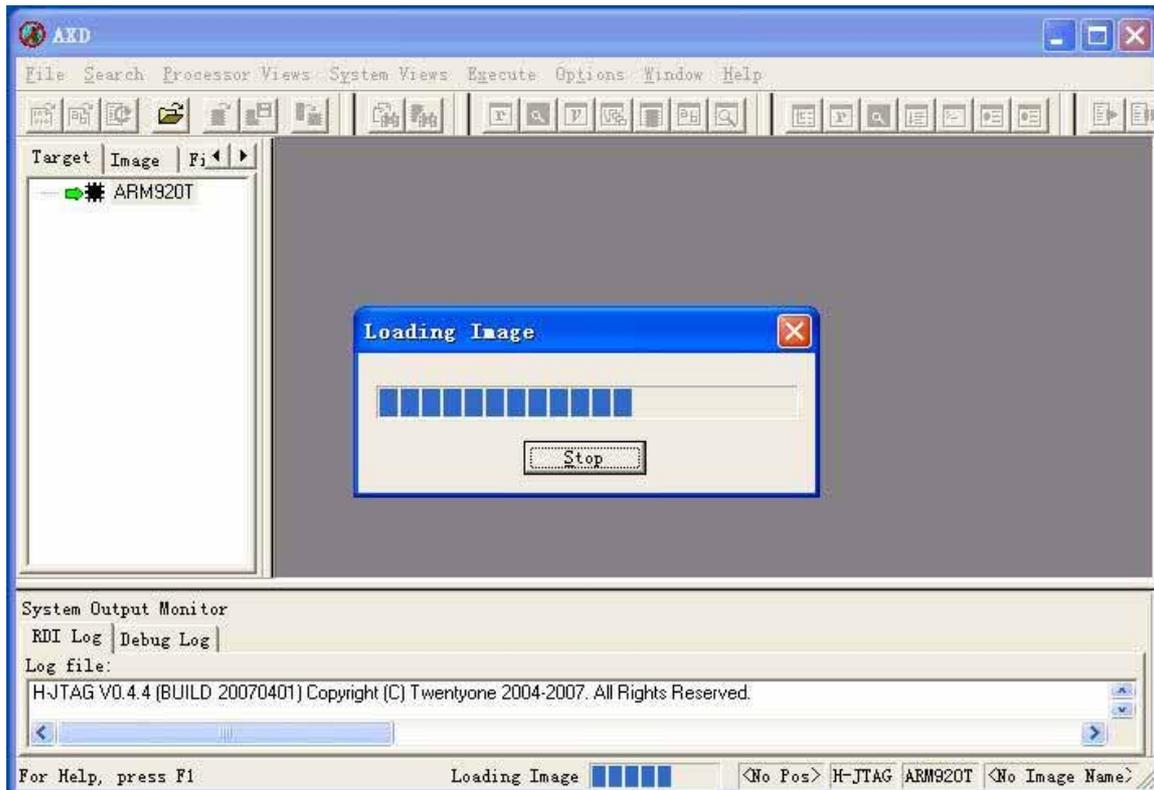
ADS1.2 の開発環境で File → Open...

2440test フォルダの 2440test.mcp を選択します。

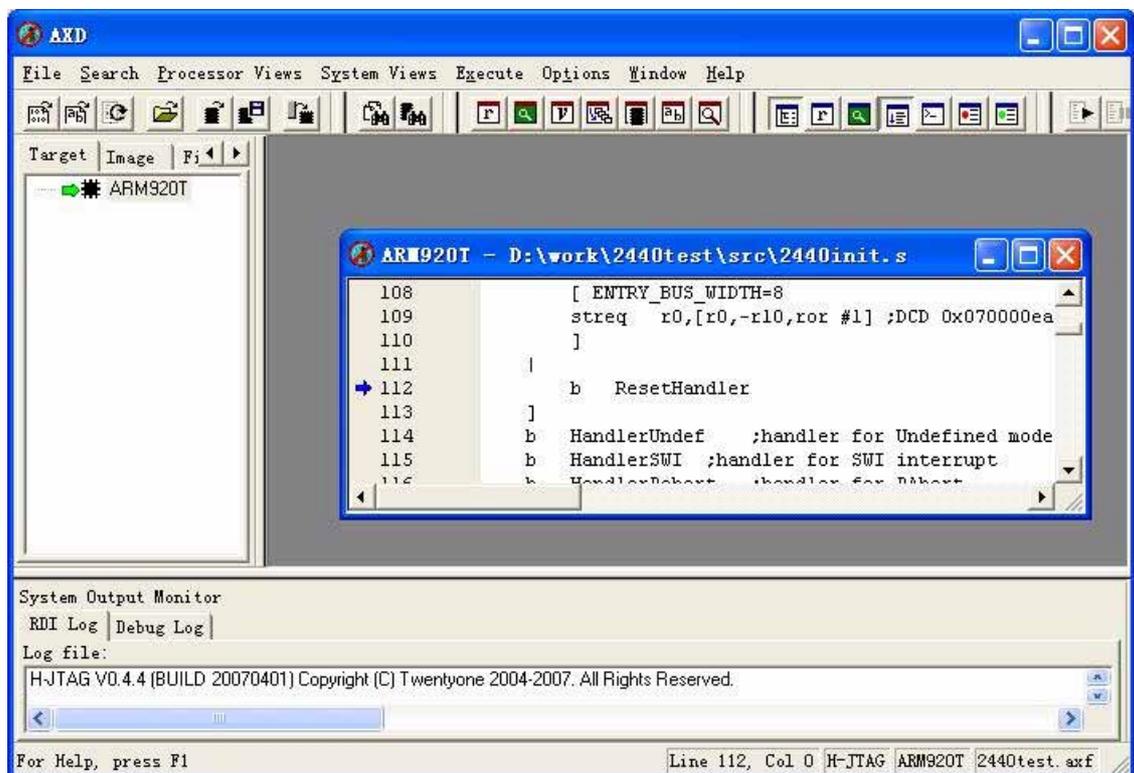


ADS1.2 のメニュー : Project → Debug

ADS1.2 は 2440test プロジェクトをコンパイルして、自動的に AXD Debugger を実行して、生成された 2440test.axf ファイルを H-JTAG で ARM9 ボードのメモリでダウンロードして、デバッグします。



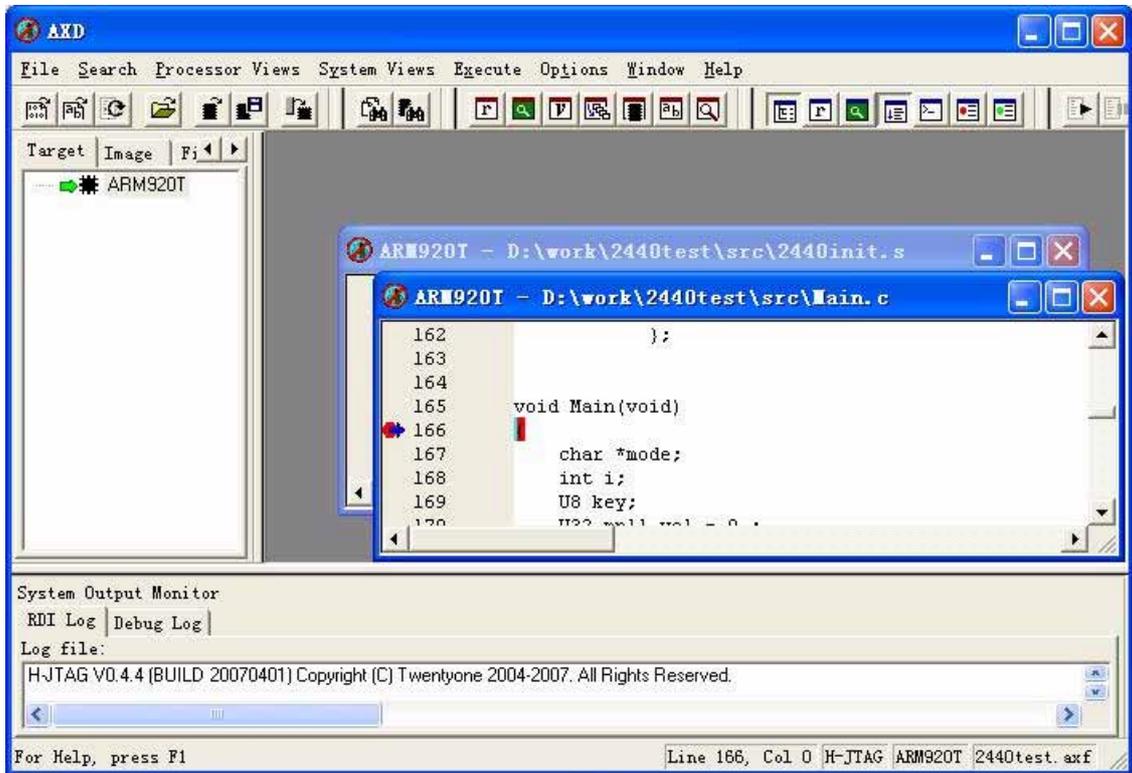
2440test.axf が大きいから、ダウンロードの時間がかかります。



ダウンロード完了の様子。

ADS1.2 のメニュー : Execute → Go

プログラムは 2440test の Main 関数に入ります。



デバッグが始まります。

付録 8 U-boot

Das U-Boot は、複数のアーキテクチャに対応したブートローダである。サポートするアーキテクチャは PPC、ARM、AVR32、MIPS、x86、68k、Nios embedded processor、MicroBlaze である。GPL のライセンスの元でリリースされている。GNU toolchain(例えば crosstool、the Embedded Linux Development Kit (ELDK)、OSELAS.Toolchain など)を使うことにより、x86 アーキテクチャのいかなるコンピュータ上でもビルドすることが出来る。

<http://www.denx.de/wiki/U-Boot/WebHome>

u-boot-1.1.6-FA24x0.tar.gz は S3C2440 用 U-boot です。以下の機能をサポートしています。

1. xmodem プロトコル
2. USB ダウンロード、パソコン側の DNW と一緒に動けます。
3. Ethernet CS8900/DM9000
4. NAND Flash の読み書き
5. Nor 又は NAND からブートできます

6. yaffs ファイルシステムの書き込み

U-boot のインストール手順 :

1. U-boot を解凍

```
#tar xvzf u-boot-1.1.6-FA24x0.tar.gz
```

2. CS8900/DM9000 の選択

include/configs/100ask24x0.hを直す

```
#if 0
```

```
#define CONFIG_DRIVER_CS8900 1 /* we have a CS8900 on-board */
```

```
#define CS8900_BASE 0x19000300
```

```
#define CS8900_BUS16 1 /* the Linux driver does accesses as shorts */
```

```
#endif
```

```
#if !defined(CONFIG_DRIVER_CS8900)
```

```
#define CONFIG_DRIVER_DM9000 1
```

```
#define CONFIG_DM9000_USE_16BIT 1
```

```
#define CONFIG_DM9000_BASE 0x20000000
```

```
#define DM9000_IO 0x20000000
```

```
#define DM9000_DATA 0x20000004
```

```
#endif
```

3. コンフィグとコンパイル

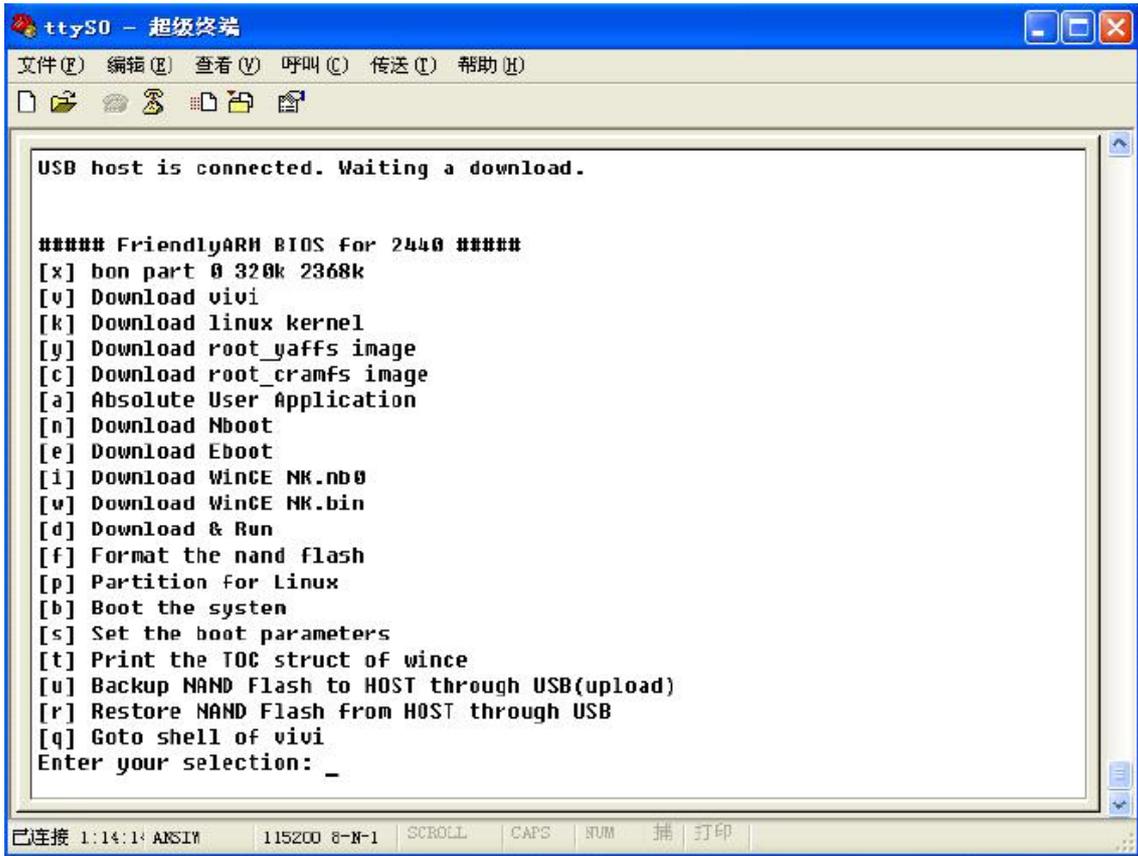
```
#make open24x0_config
```

```
#make
```

成功したら、u-boot.bin を生成します。

4. U-boot の書き込み

ARM9 を NOR Flash から起動させます。



```
USB host is connected. Waiting a download.

##### FriendlyARM BIOS for 2440 #####
[x] bon part 0 320k 2368k
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[c] Download root_cramfs image
[a] Absolute User Application
[n] Download Nboot
[e] Download Eboot
[i] Download WinCE NK.nb0
[v] Download WinCE NK.bin
[d] Download & Run
[f] Format the nand flash
[p] Partition for Linux
[b] Boot the system
[s] Set the boot parameters
[t] Print the TOC struct of wince
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
Enter your selection: _
```

メニューの中で、機能号[a]を選択して、DNW を実行します。DNW のメニュー「USBPORT」
→ 「Transmit/Restore」で生成された u-boot.bin を選択して、書き込みます。