目次

1 はじめに

本書は組み込み Linux ボード CAT シリーズ向けの 組み込み Linux ガイドです。本書をお客様のアプリケーション作成にお役立てください。本書は CAT709 / CAT760 共通のガイドブックとなっています。CAT760 については、本書で CAT709 と記述している箇所を CAT760 と、また sh3 としている箇所を sh4 と読み替えてください。

2 開発環境の構築

本機のプログラムを開発するために GNU/Linux がインストールされたPCを準備ください。現在のところ、以下の構成で動作を確認しています。

```
Debian GNU Linux Version3.0 (sarge)
```

2.1 開発キット CDROM 構成

開発キット CDROM は以下の構成です。

```
cat-cdrom/
|-- bootloader                ブートローダ(/dev/mtd0 用)バイナリ
|   |-- cat709
|   `-- cat760
|-- cross-tools               クロス開発用ツール
|   |-- debian-sarge          debian sarge 用バイナリパッケージ
|   |   |-- sh3
|   |   `-- sh4
|   `-- source
|       |-- binutils-2.15
|       |-- gcc-3.4.3-12
|       |-- gcc-3.4.3-13
|       `-- glibc-2.3.5
|-- debian-sh-base            SH3/SH4 用 DebianSarge   base.tgz
|-- etc
|-- kernel                    カーネルソースコード
|   |-- compiled-cat709
|   |-- compiled-cat760
|   `-- patch
|-- rootfs                    ルートファイルシステム
|   `-- cat709
|-- sample-driver             サンプルデバイスドライバ
|   |-- cat709port
|   |-- cat709warikomi
|   |-- hellodrv
|   `-- kmalloc
`-- sample-program            サンプルプログラム
    `-- hello
```

2.2 開発 Linux 機用 通信ソフトの準備

CAT709 はシリアルポートをコンソールとして利用します。開発PCには何かしらの通信ソフトが必要です。Windows ならば TeraTerm が便利です。ここでは Linux での通信ソフトのインストール、設定を行います。なお、後述のようにカーネルの転送に XMODEM プロトコルを用いますので XMODEM 送受信プログラム lrzsz も同時にインストールを行います。

```
# apt-get install ckermit lrzsz
```

シリアルポートデバイス（例の場合は /dev/ttyUSB0）への書き込み権限がないとシリアルポートをオープンすることができません。

```
$ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 6月 20 04:30 /dev/ttyUSB0
```

こういう表記の場合は、dialout グループのメンバーならば RW 権限がありますので、自分も dialout グループに属しましょう。スーパーユーザーになって、/etc/group ファイルをエディターで修正します。

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:ebihara
tty:x:5:
disk:x:6:
lp:x:7:lp
mail:x:8:
news:x:9:
uucp:x:10:
proxy:x:13:
kmem:x:15:
dialout:x:20:ebihara      ← dialout グループに属するユーザーを,で区切って列挙する
```

いったんログアウトしてログインしなおすと dialout グループに属しているはずです。確認してみます。

```
$ groups       ← groups コマンド実行
kaihatsu adm dialout
```

2.2.1 kermit 初期化ファイル

kermit は 個人のホームディレクトリの .kermrc 初期化ファイルを参照します。

以下の初期化ファイルを参考にしてください。~/.kermrc です。

```
set line /dev/ttyUSB0    ←（シリアルポートです /dev/ttyS0 等）
set speed 115200
set carrier-watch off
set transmit prompt 0
```

2.2.2 kermit の使い方

kermit を起動し、以下の手順で使用します。

```
$ cd /CAT709-KERNEL-DIR/arch/sh/boot
$ kermit            ← kermit 起動
C-Kermit 7.0.196, 1 Jan 2000, for Linux
 Copyright (C) 1985, 2000,
  Trustees of Columbia University in the City of New York.
Type ? or HELP for help.
C-Kermit>c                          ← c とタイプするとシリアルラインにコネクトします。
Connecting to /dev/ttyUSB0, speed 115200.
The escape character is Ctrl-¥ (ASCII 28, FS)        ←抜けるには CTRL+¥ C とタイプする
Type the escape character followed by C to get back,  ←と書いてある
or followed by ? to see other options.
----------------------------------------------------

>>                                                   ← CAT との通信画面
>>
>>boot
Loading linux...
```

Kermit を終了するには CTRL+\ 続けて C とタイプします。

2.3 クロスコンパイラのインストール

クロス開発ツールは CDROM に収録されていますが、CDROM内のツールを使用する前に Debian の以下のパッケージをインストールしてください。

```
# apt-get install gcc-3.4-base libncurses5-dev binutils-multiarch tree file hex
```

開発環境PCで以下のコマンドをタイプしてください。

CDROM のマウント

```
# mount /dev/hdc /cdrom     （但し /dev/hdc の部分は使用する PC によって異なります）
```

クロスツールのインストール

```
# cd /cdrom/cross-tools/debian-sarge/sh3        CAT760 の場合は最後が sh4 となります
# dpkg -i *.deb
```

各ツールパッケージの依存関係は以下のようになっています。個別にパッケージをインストールする際は依存関係に注意してください。

```
binutils-sh3-linux_2.15-5_i386.deb

gcc-3.4-sh3-linux_3.4.3-13_i386.deb
  libgcc1-sh3-cross_3.4.3-13_all.deb
  cpp-3.4-sh3-linux_3.4.3-13_i386.deb
```

```
g++-3.4-sh3-linux_3.4.3-13_i386.deb
  libstdc++6-dev-sh3-cross
    libstdc++6-sh3-cross
    libc6-dev-sh3-cross
      libc6-sh3-cross_2.3.5-1_all.deb
        libdb1-compat-sh3-cross_2.1.3-7_all.deb
      linux-kernel-headers-sh3-cross_2.5.999-test7-bk-17_all.deb
```

各ファイルについての説明

```
binutils            アセンブラ・リンカー
sh3-linux-gcc       gcc コンパイラ
libc6-sh3           C ライブラリ
```

2.4 NFS サーバのセットアップ

開発環境PCで以下のコマンドをタイプしてください。

NFS サーバのインストール

```
# apt-get install nfs-user-server
```

エクスポート(共有)ディレクトリの設定

```
# vi /etc/exports
```

以下ファイルの中身

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
/home       192.168.7.0/255.255.255.0(ro,no_root_squash)
```

上の例では /home 位置を、192.168.7.0/255.255.255.0 LAN に接続された全てのホストに対し、
「ReadOnly」、「root 権限でのマウントを許可」の条件で共有を許しています。

255.255.255.0 と“(”(括弧)の間にはスペースを入れてはいけません。

2.4.1 NFS サーバの再起動

```
# /etc/init.d/nfs-user-server restart
```

2.4.2 NFSクライアント側の操作方法

NFSクライアント側からは mount コマンドにて サーバの共有ディレクトリのマウントを行います。

```
# mount 192.168.7.1:/home /mnt -o ro,tcp
```

書式は

```
mount  サーバの IP アドレスもしくはホスト名:/サーバのディレクトリ   マウント先 -o オプション
オプションの例 -o ro リードオンリ、 tcp TCP 接続を利用する
```

になります。

2.5 簡単なプログラムのクロスコンパイル

以下のような簡単なプログラムを書き、クロスコンパイルを行って実行させてみましょう。

```
$ vi hello.c
```

```
#include <stdio.h>

int main(){
        int i;
        char *p;

        printf("hello sh-linux world¥n");

        // 1Mバイトメモリ確保
        p=malloc(1024*1024);

        while(1){
                printf("main=%08x, printf=%08x, malloc=%08x, i=%08x¥n",
                        main, printf, p, &i);
                sleep(1); // 1秒スリープ
        }
}
```

普通にコンパイルを行う

```
$ gcc hello.c
```

開発PCで実行

```
$ ./a.out
hello sh-linux world
main=080483f4, printf=0804831c, malloc=40158008, i=bffffaa4
main=080483f4, printf=0804831c, malloc=40158008, i=bffffaa4
main=080483f4, printf=0804831c, malloc=40158008, i=bffffaa4
main=080483f4, printf=0804831c, malloc=40158008, i=bffffaa4
main=080483f4, printf=0804831c, malloc=40158008, i=bffffaa4
:略　CTRL+C で停止します
```

gcc の出力ファイル名のデフォルトは a.out です。これは-o オプションで変更することが出来ます。また UNIX では カレントディレクトリに実行パスが通っていませんので、カレントディレクトリを示す ./ を頭に付けて実行します。

クロスコンパイルを行う

```
$ sh3-linux-gcc hello.c
```

CAT760 では

```
$ sh4-linux-gcc hello.c
```

とします。

結果を確かめる

```
$ file a.out
a.out: ELF 32-bit LSB executable, Hitachi SH, version 1 (SYSV), for GNU/Linux 2.3.99,
dynamically linked (uses shared libs), not stripped
```

開発PCで実行してみる

```
$ ./a.out
-bash: ./a.out: cannot execute binary file
```

実行できません。

コンパイルしたソフトウェアの動作

コンパイルしたプログラムを CAT 機で実行しましょう。CAT 上で以下のコマンドをタイプし、開発機のディレクトリを NFS マウントします。

```
# mount 192.168.7.1:/home /mnt -o ro,tcp
```

書式 mount IP アドレスもしくはホスト名:/共有ディレクトリ /マウント先

-o オプション ro は ReadOnly, tcp は TCP 接続を利用する(デフォルトは UDP)

コンパイルした a.out の実行

```
# cd /mnt/ebihara          (a.out ファイルのある場所に移動)
# ./a.out
hello sh-linux world
main=004005c0, printf=00400470, malloc=296aa008, i=7ba4ed58
main=004005c0, printf=00400470, malloc=296aa008, i=7ba4ed58
main=004005c0, printf=00400470, malloc=296aa008, i=7ba4ed58
main=004005c0, printf=00400470, malloc=296aa008, i=7ba4ed58
: 略 CTRL+C で停止
```

3 カーネル

カーネルはオペレーションシステムの中心部で、プロセスのスケジューリングやメモリ、IOの管理を行っています。またネットワークやファイルシステムもカーネルの仕事です。カーネルをコンフィグレーションすることで、デバイスドライバの追加削除や、読み書きできるファイルシステム形式の追加削除、ネットワーク機能の追加削除が出来ます。

3.1 カーネルの入手と展開

CDROM 内にあるカーネルを展開してください。また最新版への更新については弊社サポートWebをご覧ください。

```
$ mkdir ~/cat-kernel
$ cd ~/cat-kernel
$ tar xzvf /cdrom/kernel/linux-2.6.13.2-cat_日付.tgz
$ cd linux-2.6.13.2-cat
```

3.2 make ファイルの修正

Makefile の 194 行付近、CROSS_COMPILE の指定を修正します。CAT709 では sh3-linux-、CAT760 では sh4-linux- としてください。

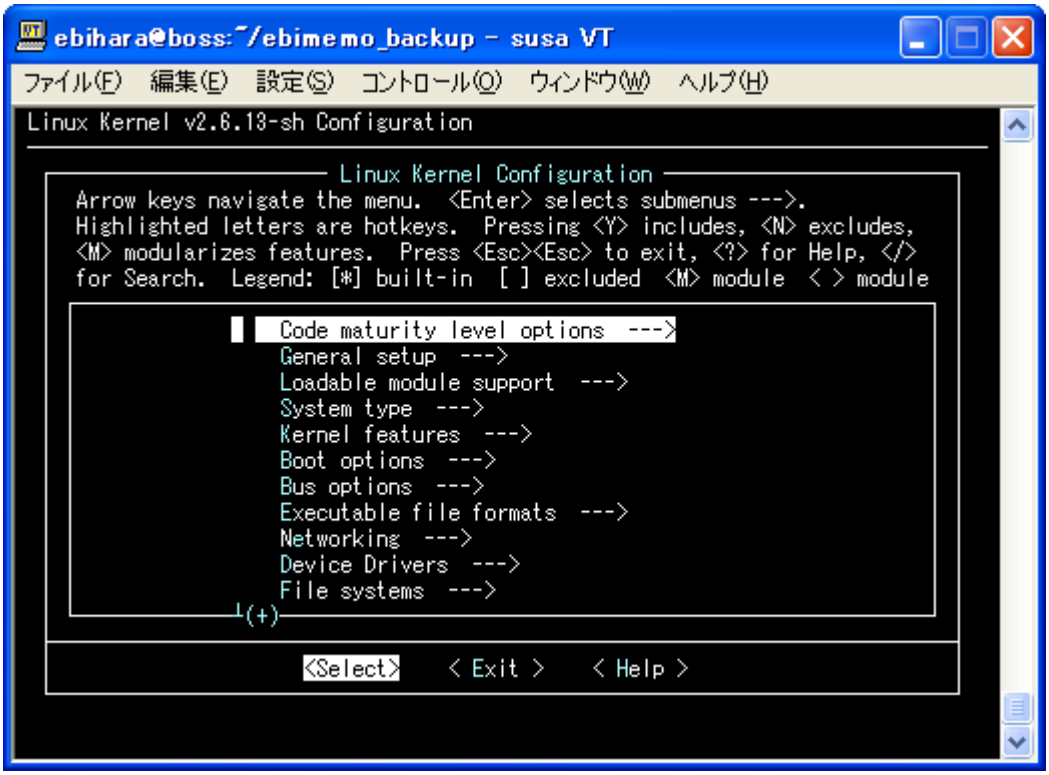
```
CROSS_COMPILE   ?= sh4-linux-
```

3.3 コンフィグレーション

以下のコマンドをタイプし、カーネルを展開してください。

```
$ cp dot-config-cat 型番  .config        CAT709/CAT760 デフォルトのコンフィグレーションをコピー
$ make menuconfig
```

ここでは上下左右のカーソル、スペースキーで選択が出来ます。<*>印はカーネルに組み込む機能、<M>印は外部モジュールファイルとしてビルドする事を示します。



コンフィグレーションは特に指定のない限り変更しないでください。

3.4 ビルド

以下のコマンドをタイプし、カーネルをビルドしてください。

```
$ make
```

コンフィグレーションした機能および開発環境PCのスペックによりますが、1GHz の機械でおよそ 5～8 分程度でコンパイルが終了します。

コンパイル後にできあがった zImage ファイルがカーネルです。zImage ファイルは以下の場所にできあがります。

```
$ ls -l arch/sh/boot/zImage
```

ROMエリアのカーネル保存領域はデフォルトで 0x130000 (1,245,184)バイトです。zImage ファイルはこのサイズ以下に収まるよう、コンフィグレーションを行ってください。なお、ROM エリアのカーネル保存領域は bootloader のメニューにて変更することが出来ます。詳しくはブートローダーの章を参照してください。

3.5 コンパイルしたカーネルのインストール

コンパイル後に出来上がった zImage ファイルをカーネル保存領域に保存します。zImage を FAT フォーマットされたコンパクトフラッシュに保存し、ブートローダメニューから以下のコマンドをタイプしてください。

```
>> cp cf0:zImage rom:kernel
```

これで zImage がカーネル保存領域に保存されました。

3.6 Linux 上での転送方法

Linux が動作中であれば zImage ファイルを /dev/mtd1 にコピーすることでカーネルのアップデートが出来ます。

```
# cp zImage /dev/mtd1
```

Linux 上で、CAT709/760 での FLASHROM は以下のデバイスとしてアクセスできます

/dev/mtd0	ブートローダ
/dev/mtd1	カーネル(zImage)
/dev/mtd2	ルートファイルシステム(rootfs)

3.7 カーネル起動パラメータ

カーネルには種々の起動パラメータがあります。起動パラメータを変更することでカーネルの動作を変更することが出来ます。ブートローダメニューの setparam コマンドで変更することが出来ます。カーネル起動パラメータは EEPROM に記録されます。最大文字数は 240 文字です。

主要なパラメータを紹介します。

```
console=ttySC0,115200      コンソール出力先の指定とボーレート（ほぼ必須）
root=0301                  root としてマウントするデバイスの指定
                           0301 はメジャー番号 03, マイナー番号 01 で/dev/hda1 のことです
                           CAT では CF スロットを意味します
                           1f02 は /dev/mtd2 を意味します
ro                         起動時にルートをリードオンリでマウントします（必須）
rootfstype=jffs2           Flash メモリをマウントするときには
                           ファイルシステム形式の指定が必要です（必須）
```

例1 CAT 内蔵 FLASH ROM(/dev/mtd2)を root としてマウントする場合（標準）

```
>> setparam console=ttySC0,115200 root=1f02 ro rootfstype=jffs2
```

例2 コンパクトフラッシュ(/dev/hda1)を root としてマウントする場合

```
>> setparam console=ttySC0,115200 root=0301 ro
```

3.8 カーネルモジュールのインストールテクニック

カーネルコンフィグレーション時にモジュールとしてビルドしたファイルのインストールには少しこつが必要です。カーネルのソースツリートップのMakefileの INSTALL_MOD_PATH を修正します。

```
#
# INSTALL_MOD_PATH specifies a prefix to MODLIB for module directory
# relocations required by build roots.  This is not defined in the
# makefile but the arguement can be passed to make if needed.
#
INSTALL_MOD_PATH=~/cat-module
```

モジュールは INSTALL_MOD_PATH で指定したディレクトリ下にインストールされます。

```
$ make modules_install
$ tree -d ~/cat-module
/home/ebihara/cat-module/
`-- lib
    `-- modules
        `-- 2.6.13-sh
            |-- build -> /home/ebihara/project/cat760/kernel/linux-2.6.13.2-cat
            |-- kernel
            |   |-- crypto
            |   |-- drivers
            |   |   |-- base
            |   |   |-- block
            |   |   |-- ide
            |   |   |-- net
            |   |   `-- pcmcia
            |   |-- fs
            |   |   |-- fat
            |   |   |-- lockd
            |   |   |-- msdos
            |   |   |-- nfs
            |   |   |-- nls
            |   |   `-- vfat
            |   `-- net
            |       |-- packet
            |       `-- sunrpc
            `-- source -> /home/ebihara/project/cat760/kernel/linux-2.6.13.2-cat
```

従って、~/cat-module/ を CAT のルートに転送します。CATにて以下のコマンドをタイプしてください。

```
# cp /NFS サーバ/ebihara/cat-module/* / -av
# depmod -a
```

4　デバイスドライバ

Linuxでは全ての機器を「ファイル」と見立てて入出力を行います。例えば/dev/ttySC0 はシリアルポート0ですが、このファイルを open()し、read()することでデータの入力を行い、write()することでデータの出力を行います。使い終わったら close()します。単純なLEDやDISPWでもデバイスドライバを書きファイルに見立てることで、アプリケーションの移植性が高まります。本章では簡単なデバイスドライバの書き方、コンパイル方法、ロード方法を説明します。

4.1　デバイスドライバ入門

Linux でのデバイスドライバを書く前に、おさえて置きべき基本的なことをまとめました。

- デバイスドライバのビルドには普通の gcc を使います。
- 特殊なパッケージは不要です。ただし カーネルソースが必要になります。
- デバイスドライバは insmod コマンドでカーネルに常駐し、rmmod コマンドで外せます。
  （DOS 時代の ADDDRV と DELDRV のようなものです。）

デバイスドライバはカーネル空間に常駐しますが、ユーザープログラム（プロセス）の『サブルーチン』みたいに呼び出されて走行するとイメージしてください。デバイスドライバの各デバイスメソッド関数はプロセスとして走行します。

カーネルコンフィグレーション時に Preempt を有効にしない限り、カーネル空間では非プリエンプション動作となります。デバイスドライバメソッドを走行中に、プロセスが勝手に切り替わることはありません。すなわち

```
while(1)
  ;
```

は際限の無い無限ループに突入して死んでしまいますが、

```
outb(上位バイトデータ, データレジスタ);
outb(下位バイトデータ, データレジスタ);
```

のような処理は必ず連続して実行され、間にかってに他のタスクに切り替わったりしないためプログラマーはある程度楽ができます。

4.1.1　簡単なデバイスドライバ

以下に最も簡単なデバイスドライバコードを載せます。単純にロードとアンロードができ、print 文でメッセージを表示します（hellodrv.c）。

```
/*
 * helldrv.c
 *   for linux-2.6 kernel
 *   written by Y.Ebihara
 *   2005/11/9  Version 1.0
 */

#include <linux/module.h>

// モジュールがロードされたとき(insmod されたとき)に呼ばれる
// 戻り値は
```

```
//   負:エラー発生
//   0 :正常終了
// とする。負を返すとモジュールは常駐しない

static int hellodrv_init(void){
  printk("hellodrv hello¥n");
  return 0;
}


// モジュールがアンロードされたときき (rmmod されたとき)に呼ばれる
// 引数も戻り値も無し

static void hellodrv_exit(void){
  printk("hellodrv goodby¥n");
}

// モジュールの初期化関数、終了関数を定義するマクロ
module_init(hellodrv_init);
module_exit(hellodrv_exit);

// モジュールライセンス定義マクロ
MODULE_LICENSE("GPL");
```

ドライバモジュールはロード時に module_init( ) マクロで定義された関数が呼ばれ、アンロード時に module_exit() マクロで定義された関数が呼ばれます。またカーネル内部に常駐するプログラムとなるためCライブラリ関数は使用できません。カーネル内部には printf の代わりとなる printk 関数が存在します。

4.1.2 ドライバのコンパイル方法

Makefile を準備します。Makefile は以下のように単純な 1 行となります。

```
obj-m   := hellodrv.o
```

コンパイルには実行機で動作しているカーネルを「ビルドしたことがある」カーネルソースツリーが必要になります。前章を参考にして、カーネルの構築を先に行っておいてください。

```
$ make -C ~/project/cat760/kernel/linux-2.6.13.2-cat/ M=`pwd` modules
```

書式は make -C <カーネルソースツリーのパス> M=`pwd` modules となります。M=の後ろは「バッククォート」記号を用いる点に注意してください。日本語キーボードでは SHIFT+7 ではなく、SHIFT+@キーになります。

実行結果

```
$ make -C ~/project/cat760/kernel/linux-2.6.13.2-cat/ M=`pwd` modules
make: Entering directory `/home/ebihara/project/cat760/kernel/linux-2.6.13.2-cat'
  CC [M]  /home/ebihara/project/cat760/cat-cdrom/sample-driver/hellodrv/hellodrv.o
  Building modules, stage 2.
  MODPOST
  CC      /home/ebihara/project/cat760/cat-cdrom/sample-driver/hellodrv/hellodrv.mod.o
  LD [M]  /home/ebihara/project/cat760/cat-cdrom/sample-driver/hellodrv/hellodrv.ko
make: Leaving directory `/home/ebihara/project/cat760/kernel/linux-2.6.13.2-cat'

$ ls
Makefile  hellodrv.BAK*  hellodrv.c*  hellodrv.ko  hellodrv.mod.c  hellodrv.mod.o  hellodrv.o
```

出来上がった *.ko がドライバファイルになります。

4.1.3　ドライバモジュールのロードとアンロード

ドライバモジュールのロードとアンロードには insmod と rmmod コマンドを使います。

```
insmod ドライバ.ko      ドライバのロード
rmmod ドライバ          ドライバのアンロード(.ko を付けない)
lsmod                   ロードされているドライバの表示
```

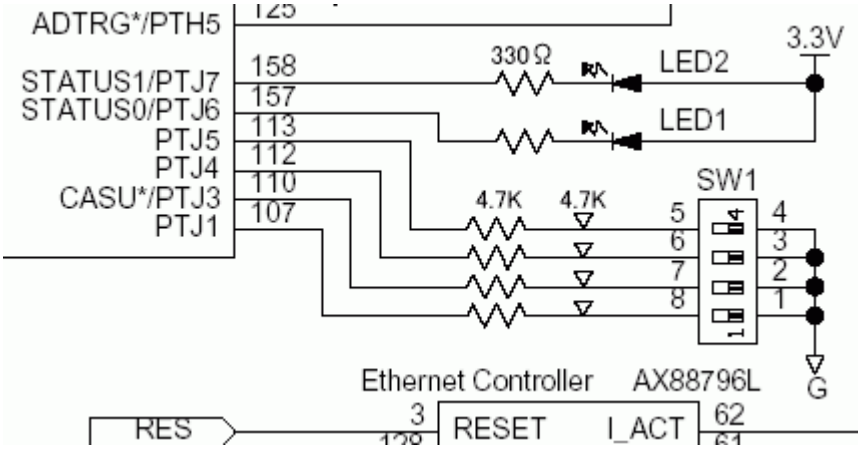
実行結果

```
# insmod hellodrv.ko
hellodrv hello
# rmmod hellodrv
hellodrv goodby
```

4.2　CAT709 DIPSW,LED 回路について

Linuxとは離れますが、CAT709 のハードウェア的なことをおさらいします。

CAT709 の LED と DIPSW はブロック回路図から以下のように抜粋できます。



LED2 個と、DIPSW 4 個が、CPU のポート J に接続されています。ポート J のアドレスは 0xA4000130 です(詳しくは SH7709 ハードウェアマニュアルに記載)。ポート J はビットごとに入出力方向を決定できます。ただし ポート J0 は RAS3L, ポート J2 は CAS3L との共有ピンになっていて、SDRAM に対する CAS3/RAS3 として使用するので、DIPSW や LED には接続されていません。LED は 0 を書き込むと点灯し、1 を書き込むと消灯します。DIPSW は ON になっているビットが 0 になります。上記の理由により J2 がとびになっているので注意してください。

	7	6	5	4	3	2	1	0
0xA4000134	LED1	LED0	SW3	SW2	SW1	x	SW0	x
	W	W	R	R	R		R	

4.2.1　メジャー番号とマイナー番号

デバイスドライバは『メジャー番号』と『マイナー番号』の2つの番号を使います。linux カーネルソースの Documentation/devices.txt に一覧表があります。メジャー番号は装置の種類を示す番号で、240～254 がローカルで使用可能です。まずはこの中から番号を選びます。今回は 241 にしました。マイナー番号は、『何枚目のカード

なのか』を識別するために使います。たとえば同じ PCI の AD コンバータカードが複数毎刺さっていたときに、何枚目のカードなのかの識別に使用します。今回は、マイナー番号0～4をDIPSW, 5～6をLED としました。

メジャー番号	マイナー番号	対応デバイス
241	0	DIPSW 0
241	1	DIPSW 1
241	2	DIPSW 2
241	3	DIPSW 3
241	4	LED 0
241	5	LED 1

各デバイスをファイルと見立てて読み書きするために、mknod コマンドを用いてデバイスファイルを作成します。

```
# rommode rw
# mknod /dev/dipsw0 c 241 0
# mknod /dev/dipsw1 c 241 1
# mknod /dev/dipsw2 c 241 2
# mknod /dev/dipsw3 c 241 3
# mknod /dev/led0 c 241 4
# mknod /dev/led1 c 241 5
# rommode ro
```

mknod コマンドの書式は以下のようになります。

```
mknod ファイル名　種類　メジャー番号　マイナー番号
```

種類は c キャラクタ型デバイスドライバ、b ブロック型デバイスドライバで、通常は c を使用します。

4.2.2　DIPSW, LEDデバイスドライバソースコード

```
/*
 * cat709port.c
 *   for linux-2.6 kernel
 *   written by Y.Ebihara
 *   2005/11/9  Version 2.0
 */
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <asm/semaphore.h>

#define DEFAULT_MAJOR_NUM   241
#define DEVNAME             "cat709port"

/* control register */
#define PCCR 0xa4000104
#define PDCR 0xa4000106
#define PLCR 0xa4000114

/* data register */
#define PCDR 0xa4000124
#define PDDR 0xa4000126
#define PJDR 0xa4000130
#define PLDR 0xa4000134
```

```
enum miscport_minor{
  DIPSW0,    // 0
  DIPSW1,    // 1
  DIPSW2,    // 2
  DIPSW3,    // 3
  LED0,      // 4
  LED1,      // 5
  MINOR_MAX
};

/* ------------------------------------------------------------ */
/* I/O access                                                   */
/* 実際の I/O 操作を行う関数                                    */
/* ------------------------------------------------------------ */

// inb(), outb() は cat709_port_map()によるアドレス変換の対象となる
// 一方、ctrl_outb(), ctrl_inb()はアドレス変換の対象とならない。
// 従って物理番地を直接書きたいときは ctrl_ を使う（速度も速い）
//   unsigned char ctrl_inb(unsigned long 物理番地)
//   ctrl_outb(unsigned char データ,  unsigned long 物理番地)

static void init_cat709port(void){
  // initialize
}


/* LED OUTPUT */
/* led = 0 or 1 */
static void cat709_led(int led, int on){
        int bit;
        unsigned char x;

        if(led==0)
                bit = 1<<6;
        else if(led==1)
                bit = 1<<7;
        else
                return;

        x=ctrl_inb(PJDR);
        if(on)
                x &= ~ bit;
        else
                x |= bit;
        ctrl_outb(x,PJDR);
}

/* DIPSW INPUT */
/* no = 0 ～ 3 */
static int cat709_dipsw(int no){
        int table[]={1,3,4,5};
        int bit;
        unsigned char d;
        if(no>3 || no<0)
                return 0;
        bit=table[no];
```

```
        d = (ctrl_inb(PJDR) >> bit ) & 1;
        return d;
}

/* ------------------------------------------------------- */
/* file operation method                                   */
/* キャラクター型デバイスドライバ                          */
/* ------------------------------------------------------- */

// 大雑把に言うと ユーザープロセスはデバイスファイルを「ファイルとみなして」
// open() し read(), write()を行う。
// デバイスメソッドは プロセスから呼び出される『サブルーチン』と思えば
// 大体あっている
// 従って open()や read(), write() は カーネル空間のテキストだが
// 『pid を持ったまま』プロセスのコンテキストとして走行する。


// このドライバをプロセスが open() したときに呼び出される

static int cat709port_open(struct inode *inode, struct file *filp){
  int minor;
  // マイナー番号取得マクロ
  minor=MINOR(inode->i_rdev);
  if(minor >= MINOR_MAX){
    printk("<1>cat709port open() error");
    return -ENODEV;
  }

  // filp->private_data は void* 型の変数で、プロセス単位(open される毎)の
  // データは private_data に保存する。
  // *filp は すべてのデバイスメソッドで引数としてやってくるので
  // filp->private_data はすべてのデバイスメソッドから参照することができる。
  // 変数がひとつで足りないときは 構造体の形をあらかじめ定義しておいて
  // open()時に kmalloc() で構造体の実体を作り、ポインタを private_data に格納する。
  // もちろん デバイスを close() するときに kfree()することを忘れないように
  filp->private_data = (void*)minor;
  return 0;
}

static int cat709port_close(struct inode *inode, struct file *filp){
  return 0;
}

static int cat709port_read(struct file *filp, char *user_buf,size_t count,loff_t *ppos){
  int minor;
  minor=(int)(filp->private_data);

  if( minor>=DIPSW0 && minor<=DIPSW3){
        unsigned char buffer[255];  // スタックサイズ=ローカル変数のサイズ=約 7kbyte 程度なので注
意
        int len,data,dipswno;
        dipswno = minor-DIPSW0;
        data = cat709_dipsw(dipswno);
        // 文字列に変換する
        len=sprintf(buffer,"%d¥n",data);
        // copy_to_user(転送先,転送元,バイト長) で
```

```
        // ユーザープロセスにデータを返す
        copy_to_user(user_buf,buffer,len);
        return len;  // 転送バイト数を返す
  }
  return 0;
  // 戻り値は
  //   負:エラー
  //   0 :end-of-file に到達
  //   正:転送したバイト数
  // としなければならない
}


static int cat709port_write(struct file *filp, const char *user_buf,size_t count,loff_t *ppos){
  int minor;
  minor=(int)(filp->private_data);

  if( minor>=LED0 && minor<=LED1){
        unsigned char buffer[255];  // スタックサイズ=ローカル変数のサイズ=約 7kbyte 程度なので注
意
        int ledno;

        ledno=minor-LED0;
        copy_from_user(buffer,user_buf,1);
        if(buffer[0]=='0')
                cat709_led(ledno, 0);
        else if(buffer[0]=='1')
                cat709_led(ledno, 1);
        return count;
  }
  return count;
  // 戻り値は
  //   負:エラー
  //   0 :end-of-file に到達
  //   正:転送したバイト数
  // としなければならない
  // write()の時は、ユーザープロセスが書き出ししたかったバイト数(count)を
  // そのまま返すと、ユーザープロセスは大満足する(^-^)
}

// file_operations 構造体とはジャンプテーブルやベクターテーブルのようなもので
// デバイスファイルを open() read() write() close() されたときに呼び出す
// 関数一覧である

static struct file_operations cat709port_fops = {
  owner:THIS_MODULE,            // おまじない
  read:cat709port_read,         // read
  write:cat709port_write,       // write
  open:cat709port_open,         // open
  release:cat709port_close      // close
};


/* ------------------------------------------------------- */
/* driver module load / unload */
/* ------------------------------------------------------- */
```

```
// モジュールがロードされたとき(insmod されたとき)は
// init_module()が呼ばれる
// 戻り値は
//   負:エラー発生
//   0 :正常終了
// とする。負を返すとモジュールは常駐しない

static int cat709port_initmodule(void){
  printk("<1>cat709port hello¥n");

  // register_chrdev(メジャー番号, デバイス名,
  //                 ファイルオペレーションズ構造体へのポインタ)
  // で、カーネルに対してキャラクタ型デバイスドライバの登録を行う
  if(register_chrdev(DEFAULT_MAJOR_NUM,DEVNAME,&cat709port_fops) != 0){
    printk("<1>cat709port register fail¥n");
    return -1;
  }
  // ハードウェア的な初期化
  init_cat709port();
  return 0;
}

// モジュールがアンロードされたときき (rmmod されたとき)は
// cleanup_module()が呼び出される

static void cat709port_exitmodule(void){
  printk("<1>cat709port goodby¥n");
  unregister_chrdev(DEFAULT_MAJOR_NUM,DEVNAME);
}

module_init(cat709port_initmodule);
module_exit(cat709port_exitmodule);

MODULE_LICENSE("GPL");
```

以下のようなMakefileを用意してコンパイルし、ロードしてみましょう。

```
obj-m   := cat709port.o
```

コンパイル

```
$ make -C ~/project/cat760/kernel/linux-2.6.13.2-cat/ M=`pwd` modules
make: Entering directory `/home/ebihara/project/cat760/kernel/linux-2.6.13.2-cat'
  CC [M]  /home/ebihara/project/cat760/cat-cdrom/sample-driver/cat709port/cat709port.o
  Building modules, stage 2.
  MODPOST
  CC      /home/ebihara/project/cat760/cat-cdrom/sample-driver/cat709port/cat709port.mod.o
  LD [M]  /home/ebihara/project/cat760/cat-cdrom/sample-driver/cat709port/cat709port.ko
make: Leaving directory `/home/ebihara/project/cat760/kernel/linux-2.6.13.2-cat'
```

確認

```
$ ls
Makefile  cat709port.c  cat709port.ko  cat709port.mod.c  cat709port.mod.o  cat709port.o
```

4.2.3　CAT709 DIPSW, LEDドライバのロードと実行

ロードと実行はCAT709で行います。NFS を利用して開発機のディレクトリに移動し、insmod コマンドでドライバをロードします。

```
# mount 開発機:/ /mnt
# cd /mnt/どこかのディレクトリ
# insmod cat709port.ko
cat709port hello
```

組み込まれたか確認します

```
#  lsmod
module                  Size  Used by
cat709port              1984  0              ←これがドライバ
pcmcia_core            30464  0
nfs                    88124  1
lockd                  57028  2 nfs
sunrpc                124400  3 nfs,lockd
ne                      5428  0
8390                    8304  1 ne
```

メジャー番号の確認

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
128 ptm
136 pts
204 ttySC
241 cat709port          ←これがドライバ

Block devices:
 31 mtdblock
```

ドライバに対してデータの出力(LEDの点灯、消灯)

```
# echo 1 > /dev/led0       LED0 に対して 1 を出力。LED0 点灯
# echo 0 > /dev/led0         LED0 消灯
# echo 1 > /dev/led1         LED1 点灯
# echo 0 > /dev/led1         LED0 消灯
```

同様に、DIPSW を読み込みます。

```
# cat /dev/dipsw0
```

```
0
0
0
0
0
0
0
:  略  CTRL+C で終了
```

4.3 完成したドライバの組み込み

完成したドライバを CAT709 の FlashRom に組み込みます。ドライバは

```
/lib/modules/カーネルバージョン/drivers/misc
```

ディレクトリの中にインストールします。

```
# mkdir /lib/modules/2.6.13-sh/kernel/drivers/misc
# rommode rw
# cp cat709port.ko /lib/modules/2.6.13-sh/kernel/drivers/misc/
# depmod -a
```

ドライバモジュール(*.ko)を追加削除した場合は depmod -a を一度実行します。

4.4 モジュールの自動ロード

起動時に自動的にドライバが読み込まれるようにするには、/etc/modules ファイルに記述します。

```
# vi /etc/modules
```

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
ne io=0x300
nfs
cat709port              ←1 行追加
```

5 作成したソフトの自動起動

LED ドライバが組み込めましたので以下のようなプログラムを作り、LED を ON/OFF させましょう。またこのプログラムを /usr/local/bin に led_control という名前で保存します。(led_control.c)

```
/*
 * led_control.c
 *    CAT709 LED control program
 *    2005.11.09 Y.Ebihara
 */

#include <stdio.h>
#include <fcntl.h>

#define DEVNAME "/dev/led0"

int main(){
        int fd;
```

```
        char buffer[1];
        fd=open(DEVNAME,O_RDWR);
        if(fd<0){
                perror("");
                exit(1);
        }
        while(1){
                printf("点灯¥n");
                buffer[0]='1';
                write(fd,buffer,1);
                sleep(1); // 1秒スリープ

                printf("消灯¥n");
                buffer[0]='0';
                write(fd,buffer,1);
                sleep(1);
        } // 無限ループ
}
```

コンパイル（開発機にて）

```
$ sh3-linux-gcc -O2 led_control.c -o led_control
```

実行（CAT709 にて）

```
# ./led_control
消灯
点灯
消灯
点灯
```

/usr/local/bin に保存します

```
# rommode rw
# cp led_control /usr/local/bin/
# rommode ro
```

5.1 スタートストップ スクリプト

作成した led_control プログラムを「デーモン」として動作させるために、スタートストップスクリプトを準備します。
/etc/init.d/ に skeleton がありますので、これをコピーして修正します。

```
# rommode rw
# cd /etc/init.d/
# cp skeleton led_control
# vi led_control
```

スクリプトを次のように修正します(led_control)

```
DESC="LED Control"                      ← 説明文
NAME=led_control                        ← コマンド名
DAEMON=/usr/local/bin/$NAME             ← パス /usr/local/bin に修正
PIDFILE=/var/run/$NAME.pid
```

```
SCRIPTNAME=/etc/init.d/$NAME

# Gracefully exit if the package has been removed.
test -x $DAEMON || exit 0

# Read config file if it is present.
#if [ -r /etc/default/$NAME ]
#then
#       . /etc/default/$NAME
#fi

#
#       Function that starts the daemon/service.
#
d_start() {                         ↓ -start の後ろに -b -m を追加
        start-stop-daemon --start -b -m --quiet --pidfile $PIDFILE ¥
                --exec $DAEMON
}
以下変更無し
```

修正が終わりましたら、start 及び stop の引数で、プログラムの実行、終了が出来るか確認します。

```
# /etc/init.d/led_control start
Starting LED Control: led_control.
# /etc/init.d/led_control stop
Stopping LED Control: led_control.
```

起動時に自動的に start するためには、/etc/rc2.d ディレクトリからシンボリックリンクを張ります。

```
supercat:~# cd /etc/rc2.d/
supercat:/etc/rc2.d# ls
S10sysklogd  S14ppp      S20boa    S20makedev  S89atd   S99rmnologin
S11klogd     S18portmap  S20inetd  S20pcmcia   S89cron  S99stop-bootlogd
supercat:/etc/rc2.d# ln -s ../init.d/led_control S90led_control
```

このディレクトリは「ランレベル２」になったときにファイル名順に実行されるため、S?? の部分は行番号のと考えることも出来ます。今回は S90 としました。

最後に rommode に戻します。

```
# rommode ro
```

電源を再投入し、LED が点滅することを確認してください。

6　デバイスドライバの高度なプログラミング

本章ではより実践的なプログラミング学習のため、高度な知識について取得します。

6.1 メモリの確保解放

デバイスドライバ中からメモリを確保、解放するには kmalloc() あるいは vmalloc() を使用します。どちらもCライブラリの malloc()と使い方は同じです。

```
/*
 * kmalloc.c
 *   for linux-2.6 kernel
 *   written by Y.Ebihara
 *   2005/11/9  Version 1.0
 */

#include <linux/module.h>
#include <linux/slab.h>
#include <linux/vmalloc.h>

#define SIZE 1024*16   // 16Kbyte

char *kbuffer;
char *vbuffer;

static int kmalloc_init(void){
        kbuffer = kmalloc(SIZE, GFP_KERNEL);
        vbuffer = vmalloc(SIZE);
        printk("kmalloc hello kbuffer=%lx vbuffer=%lx¥n",
                (unsigned long)kbuffer,(unsigned long)vbuffer);
        return 0;
}

static void kmalloc_exit(void){
        kfree(kbuffer);
        vfree(vbuffer);
        printk("kmalloc goodby¥n");
}

// モジュールの初期化関数、終了関数を定義するマクロ
module_init(kmalloc_init);
module_exit(kmalloc_exit);

// モジュールライセンス定義マクロ
MODULE_LICENSE("GPL");
```

kmalloc()は「物理メモリが連続している」事を保証する代わりにサイズが 128K バイトまでと制限されています。128K バイト以下でも連続したメモリ空間が見つからない場合は失敗する可能性があります。vmalloc()は仮想メモリ空間的に連続されているメモリを確保します。物理メモリは非連続かもしれません。従って、DMA アクセスや画像フレームバッファなど、ハードウェアが連続して読み書きするバッファとしては vmalloc()は使用できません。

リファレンス

```
メモリの確保
        kbuffer = kmalloc(SIZE, GFP_KERNEL);
        vbuffer = vmalloc(SIZE);
メモリの解放
        kfree(kbuffer);
```

```
        vfree(vbuffer);
```

6.2 割り込み

Linuxはカーネルが割り込みの管理をしますのでユーザープログラムは容易に割り込みハンドラを作ることが出来ます。以下の例は SH3 CPU内蔵タイマ1使い、毎秒10回の割り込みを発生させるプログラムです。割り込みハンドラではLED1の点灯・消灯を反転しています。(cat709warikomi.c)

```
/*
 * cat709warikomi.c
 *   for linux-2.6 kernel
 *   written by Y.Ebihara
 *   2005/11/9  Version 2.0
 */
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/interrupt.h>
#include <asm/io.h>
#include <asm/uaccess.h>

#define DEVNAME "warikomi"

#define PJDR 0xa4000130

/* LED OUTPUT */
/* led = 0 or 1 */
static void cat709_led(int led, int on){
        int bit;
        unsigned char x;

        if(led==0)              bit = 1<<6;
        else if(led==1)         bit = 1<<7;
        else                    return;

        x=ctrl_inb(PJDR);

        if(on)                  x &= ~ bit;
        else                    x |= bit;

        ctrl_outb(x,PJDR);
}

/* ------------------------------------------------------- */
/* SH7709 内蔵 Timer1 操作                                  */
/* ------------------------------------------------------- */

/* ---- timer controler --- */

#define TOCR  0xffffffe90  /* byte  */
#define TSTR  0xffffffe92  /* byte  */

/* ch 1 */
#define TCOR1 0xffffffea0  /* long  */
#define TCNT1 0xffffffea4  /* long  */
```

```
#define TCR1  0xfffffea8  /* short */

#define MODULE_CLK      29421200
#define TIMER_CLK       (MODULE_CLK/16)      /* timer1に入力されるクロック*/
#define TIMER_HZ        10

#define TMU1_IRQ 17

void tmu1_init(void){
        unsigned long interval;
        interval = TIMER_CLK / TIMER_HZ;

        /* タイマ１停止 */
        ctrl_outb(ctrl_inb(TSTR)&~0x2,TSTR); /* ch1 カウントダウン停止 */

        /* タイマ１スタート */
        ctrl_outw(0x0021, TCR1);             /* 割り込み有効、プリスケーラ=1/16 */
        ctrl_outl(interval, TCOR1);          /* リロードレジスタにインターバルセット */
        ctrl_outl(interval, TCNT1);          /* カウントダウンレジスタにインターバルセット */
        ctrl_outb(ctrl_inb(TSTR)|0x2, TSTR); /* ch1 カウントダウン開始 */
}

void tmu1_stop(void){
        ctrl_outb(ctrl_inb(TSTR)&~0x2,TSTR); /* ch1 カウントダウン停止 */
}


/* ------------------------------------------------------- */
/* 割り込みルーチン                                        */
/* ------------------------------------------------------- */

static irqreturn_t tmu1handler(int irq, void *dev, struct pt_regs *regs)
{
        static int flag=0;

        /* タイマー１割り込みフラグリセット */
        ctrl_outw(0x0021, TCR1);

        cat709_led(1,flag);
        flag=1-flag;

        return IRQ_HANDLED;
}


/* ------------------------------------------------------- */
/* driver module load / unload */
/* ------------------------------------------------------- */

// モジュールがロードされたとき(insmod されたとき)は
// init_module()が呼ばれる
// 戻り値は
//   負:エラー発生
//   0 :正常終了
// とする。負を返すとモジュールは常駐しない
```

```
static int cat709warikomi_initmodule(void){
        printk("<1>cat709warikomi hello¥n");

        // 割り込みルーチンの登録
        request_irq(TMU1_IRQ,tmu1handler,0,DEVNAME,0);    /* 割り込みルーチン登録 */

        // ハードウェア的な初期化
        tmu1_init();
        return 0;
}

// モジュールがアンロードされたときき (rmmod されたとき)は
// cleanup_module()が呼び出される

static void cat709warikomi_exitmodule(void){
        printk("<1>cat709warikomi goodby¥n");

        tmu1_stop();
        free_irq(TMU1_IRQ,NULL);
}

module_init(cat709warikomi_initmodule);
module_exit(cat709warikomi_exitmodule);

MODULE_LICENSE("GPL");
```

リファレンスは以下の通りです。

割り込み関数

```
static irqreturn_t tmu1handler(int irq, void *dev, struct pt_regs *regs)
{
        return IRQ_HANDLED;  // 割り込みを正常にキャッチしたら IRQ_HANDLED を返す
}
```

割り込みハンドラの登録関数

```
request_irq(IRQ 番号, 割り込み関数, フラグ , デバイス名, void*シグネチャ);
```

割り込みハンドラの解放関数

```
free_irq(IRQ 番号, void*シグネチャ);
```

6.3 プロセスの停止、再開

例えばエディターはキー入力が発生するまで静かに待ちます。このように入力や出力が発生するまでタスクを待たせるのはデバイスドライバの仕事です。この例では read に入ってきたプロセスを割り込み 100 回発生するまで待たせる例を示します。

```
/*
 * cat709warikomi2.c
 *   for linux-2.6 kernel
 *   written by Y.Ebihara
 *   2005/11/9  Version 2.0
 */
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
```

```
#include <linux/interrupt.h>
#include <linux/sched.h>
#include <asm/io.h>
#include <asm/uaccess.h>

#define DEFAULT_MAJOR_NUM  242
#define MINOR_MAX          0

#define DEVNAME "warikomi"


#define PJDR 0xa4000130

wait_queue_head_t my_stop_point; // 待ち列

/* LED OUTPUT */
/* led = 0 or 1 */
static void cat709_led(int led, int on){
        int bit;
        unsigned char x;

        if(led==0)              bit = 1<<6;
        else if(led==1)         bit = 1<<7;
        else                    return;

        x=ctrl_inb(PJDR);

        if(on)                  x &= ~ bit;
        else                    x |= bit;

        ctrl_outb(x,PJDR);
}


/* ------------------------------------------------------ */
/* SH7709 内蔵 Timer1 操作                                 */
/* ------------------------------------------------------ */

/* ---- timer controler --- */

#define TOCR  0xfffffe90  /* byte  */
#define TSTR  0xfffffe92  /* byte  */

/* ch 1 */
#define TCOR1 0xfffffea0  /* long  */
#define TCNT1 0xfffffea4  /* long  */
#define TCR1  0xfffffea8  /* short */

#define MODULE_CLK      29421200
#define TIMER_CLK       (MODULE_CLK/16)      /* timer1に入力されるクロック*/
#define TIMER_HZ        10

#define TMU1_IRQ 17

void tmu1_init(void){
        unsigned long interval;
```

```
        interval = TIMER_CLK / TIMER_HZ;

        /* タイマ１停止 */
        ctrl_outb(ctrl_inb(TSTR)&~0x2,TSTR); /* ch1 カウントダウン停止 */

        /* タイマ１スタート */
        ctrl_outw(0x0021, TCR1);             /* 割り込み有効、プリスケーラ=1/16 */
        ctrl_outl(interval, TCOR1);          /* リロードレジスタにインターバルセット */
        ctrl_outl(interval, TCNT1);          /* カウントダウンレジスタにインターバルセット */
        ctrl_outb(ctrl_inb(TSTR)|0x2, TSTR); /* ch1 カウントダウン開始 */
}

void tmu1_stop(void){
        ctrl_outb(ctrl_inb(TSTR)&~0x2,TSTR); /* ch1 カウントダウン停止 */
}


/* ------------------------------------------------------- */
/* 割り込みルーチン                                        */
/* ------------------------------------------------------- */

static irqreturn_t tmu1handler(int irq, void *dev, struct pt_regs *regs)
{
        static int flag=0;

        /* タイマー１割り込みフラグリセット */
        ctrl_outw(0x0021, TCR1);

        // LED 点滅
        cat709_led(1,flag);
        flag=1-flag;

         // プロセス起床
        wake_up_interruptible(&my_stop_point);

        return IRQ_HANDLED;
}

/* ------------------------------------------------------- */
/* デバイスメソッド                                        */
/* ------------------------------------------------------- */

int opens=0;

static int cat709warikomi2_open(struct inode *inode, struct file *filp){
        int minor;
        int countdown;
// マイナー番号取得マクロ
        minor=MINOR(inode->i_rdev);
        if(minor > MINOR_MAX){
                printk("<1>cat709warikomi2_open open() error");
                return -ENODEV;
        }

        // open されている回数
        opens++;
```

```
        // 割り込みの有効
        enable_irq(TMU1_IRQ);

        // カウントダウン値
        countdown=100;
        filp->private_data = (void*)countdown;

        return 0;
}

static int cat709warikomi2_close(struct inode *inode, struct file *filp){
        opens--;
        if(opens==0){
                // 割り込みの無効
                disable_irq(TMU1_IRQ);
        }
        return 0;
}

static int cat709warikomi2_read(struct file *filp, char *user_buf,size_t count,loff_t *ppos){
        int countdown;
        countdown = (int)filp->private_data;

        while(countdown > 0){
                // 割り込みがするまでプロセスは休眠に入る
                interruptible_sleep_on(&my_stop_point);
                countdown --;
        }

        return 0;
}



// file_operations 構造体とはジャンプテーブルやベクターテーブルのようなもので
// デバイスファイルを open() read() write() close() されたときに呼び出す
// 関数一覧である

static struct file_operations cat709warikomi2_fops = {
  owner:THIS_MODULE,            // おまじない
  read:cat709warikomi2_read,        // read
  open:cat709warikomi2_open,        // open
  release:cat709warikomi2_close     // close
};

/* ---------------------------------------------------------- */
/* driver module load / unload */
/* ---------------------------------------------------------- */

// モジュールがロードされたとき(insmod されたとき)は
// init_module()が呼ばれる
// 戻り値は
//   負:エラー発生
//   0 :正常終了
// とする。負を返すとモジュールは常駐しない
```

```
static int cat709warikomi2_initmodule(void){
        printk("<1>cat709warikomi2 hello¥n");

        init_waitqueue_head(&my_stop_point); // 待ち列の初期化

        // キャラクタ型デバイスドライバの登録
        register_chrdev(DEFAULT_MAJOR_NUM,DEVNAME,&cat709warikomi2_fops);

        // 割り込みルーチンの登録
        request_irq(TMU1_IRQ,tmu1handler,0,DEVNAME,0);    /* 割り込みルーチン登録 */

        // 割り込みの無効
        disable_irq(TMU1_IRQ);

        // ハードウェア的な初期化
        tmu1_init();


        return 0;
}

// モジュールがアンロードされたときき (rmmod されたとき)は
// cleanup_module()が呼び出される

static void cat709warikomi2_exitmodule(void){
        printk("<1>cat709warikomi2 goodby¥n");
        tmu1_stop();
        free_irq(TMU1_IRQ,NULL);
        unregister_chrdev(DEFAULT_MAJOR_NUM,DEVNAME);
}

module_init(cat709warikomi2_initmodule);
module_exit(cat709warikomi2_exitmodule);

MODULE_LICENSE("GPL");
```

プロセスを休眠させるには「待ち列」を使用します。

リファレンス

インクルードファイル

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

待ち列変数

```
wait_queue_head_t my_stop_point;
```

待ち列の初期化

```
init_waitqueue_head(&my_stop_point);
```

休眠に入る

```
interruptible_sleep_on(&my_stop_point);
```

起こす

```
wake_up_interruptible(&my_stop_point);
```

7　Debian SH を使った本格システムの構築

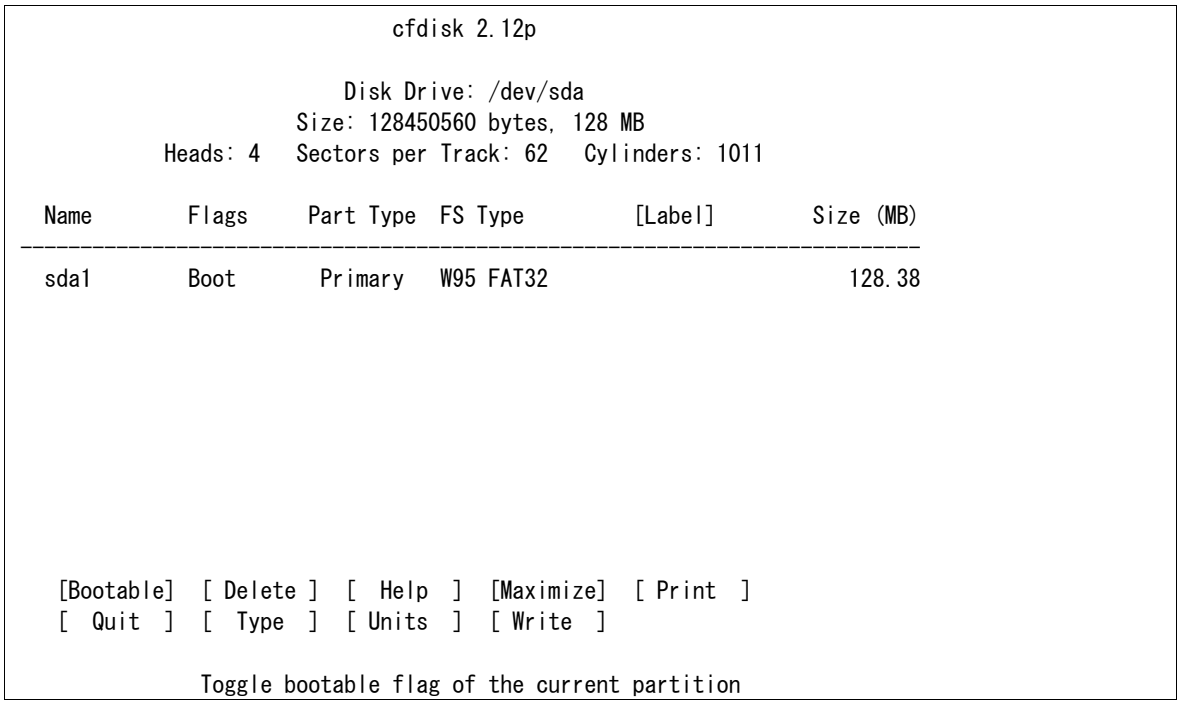
CDROM 内に Debian Sarge システムの SH 版を用意しました。コンパクトフラッシュ、またはマイクロドライブを使い、本格的なオペレーションシステムを構築することが出来ます。小型、省電力でありながら本格的な UNIX オペレーティングシステムを構築し、PCの置き換えとして使用することが出来ます。

用意する物　128MByte 以上のコンパクトフラッシュ（2G 以上のマイクロドライブ奨励）

7.1　コンパクトフラッシュのフォーマット

開発機にコンパクトフラッシュを装着しパーテーションを作ります。

```
$ su-
Passswd:
# cfdisk /dev/sda
ただし環境によってコンパクトフラッシュを認識するデバイスが異なります。（/dev/hde 等）
```



新品のコンパクトフラッシュの場合は FAT32 パーテーションがありますので削除し、新たに Linux パーテーションを作ります。キーボードの矢印キーでカーソル移動、「ENTER」キーで決定です。

1. [Delete]
2. [New] → [Primary] → Size(in MB):128
3. [Write] → Are you sure (yes or no): yes
4. [Quit]

コンパクトフラッシュを ext2 形式でフォーマットします。

```
# mkfs.ext2 /dev/sda1                    /dev/sda1 の部分は環境によって異なります。
```

マウントします

```
# mkdir /mnt/cf                          マウントポジションの作成
```

```
# mount /dev/sda1 /mnt/cf              マウント
# ls /mnt/cf
lost+found
```

7.2 Debian SH ベースの展開

CDROM内に Debian Sh のベースファイル一式がありますので展開します。

```
# cd /mnt/cf                                                    コンパクトフラッシュに移動
#  tar xzvf /cdrom/debian-sh-base/base-sarge-20051108-sh3.tgz   ベースの展開
bin/                                                           CAT760 の場合は -sh4 を
bin/bash                                                       使用してください。
:  後略
```

7.3 設定ファイルの記述

/etc ディレクトリ以下にある基本的な設定ファイルを記述します。

ログインを管理する getty の設定

```
# vi /mnt/cf/etc/inittab
```

```
1:2345:respawn:/sbin/getty 115200 ttySC0          ←115200bps, ttySC0 とする
#2:23:respawn:/sbin/getty 38400 tty2              ←以下はコメントアウト
#3:23:respawn:/sbin/getty 38400 tty3
#4:23:respawn:/sbin/getty 38400 tty4
#5:23:respawn:/sbin/getty 38400 tty5
#6:23:respawn:/sbin/getty 38400 tty6
```

root でのログインを許す端末

```
# vi /mnt/cf/etc/securetty
```

```
# for people with serial port consoles
ttySC0
```

ホスト名の設定。ここではホスト名を supercat としています。

```
# echo "supercat" > /mnt/cf/etc/hostname
```

ルートファイルシステムを確認します。

```
# vi /mnt/cf/etc/fstab
```

```
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
/dev/hda1       /               ext2    defaults,errors=remount-ro   0 1
proc            /proc           proc    defaults        0 0
```

IP アドレス関連の設定を行います。

```
# vi /mnt/cf/etc/network/interfaces
```

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
# The loopback interface
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
        address 192.168.7.100
        netmask 255.255.255.0
        gateway 192.168.7.1
```

基本的な hosts ファイルの設定をします。

```
# vi /mnt/cf/etc/hosts
```

```
127.0.0.1       localhost.localdomain   localhost
192.168.7.133   supercat
```

ネームサーバおよびデフォルトドメイン

```
# vi /mnt/cf/etc/resolv.conf
```

```
nameserver 192.168.7.1
search yourdomain.dom
```

apt ライン

```
# vi /mnt/cf/etc/apt/sources.list
```

```
deb http://www.si-linux.co.jp/pub/debian-sh/ sarge main non-free contrib
deb-src http://ftp.debian.org/debian/ sarge main non-free contrib
deb-src http://ftp.debian.or.jp/debian/ sarge main non-free contrib
deb-src http://ftp.debian.or.jp/debian-non-US sarge/non-US main contrib non-free
```

以上の設定を行い、アンマウントしてコンパクトフラッシュを抜きます。

```
# cd                カレントディレクトリがコンパクトフラッシュ上だと busy でアンマウントできない
# umount /mnt/cf
# exit              root ユーザーでの操作を終えます
$
```

7.4　カーネル再構築

コンパクトフラッシュを ／（ルート）としてマウントするため、IDE ドライバと ext2 ファイルシステムをカーネルにスタティックに組み込む必要が亜あります。第3章カーネルを参考にし、開発機でカーネルを再構築してください。あらかじめコンフィグレーションを行った設定ファイルを、dot-config-cat709-ide として用意しました。

```
$ cd /カーネルを展開したディレクトリ
```

```
$ cp dot-config-cat 型番  .config
$ make menuconfig
```

```
Device Drivers  --->
  ATA/ATAPI/MFM/RLL support  --->
    <*> ATA/ATAPI/MFM/RLL support
    <*>   Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy support
    <*>     Include IDE/ATA-2 DISK support
    <*>     generic/default IDE chipset support
File systems  --->
  <*> Second extended fs support
        ↑コンパクトフラッシュが ext2 形式ですから、ext2fs の組み込みは必須です。
  Miscellaneous filesystems  --->
    <M> Journalling Flash File System v2 (JFFS2) support
        ↑本当は組み込みたいところですが、カーネルサイズがオーバーするのでモジュールとします。
```

ビルド

```
$ make clean
$ make
```

ファイルサイズを確認します。

```
$ ls -l arch/sh/boot/zImage
```

3章を参考に、zImage を CAT に書き込んでください。

7.4.1　モジュールについて補足

モジュールとしてコンパイルしたファイルをコンパクトフラッシュにインストールする必要がありますので、再度コンパクトフラッシュを /mnt/cf にマウントし、Makefile を書き換えてインストールします。以下に手順を記載します。

```
$ vi Makefile
```

```
INSTALL_MOD_PATH=/mnt/cf
```

```
$ su                          root ユーザーになる
#  mount /dev/sda1 /mnt/cf    コンパクトフラッシュを/mnt/cf にマウントする
#  make modules_install       モジュールを INSTALL_MOD_PATH にインストールする
#  umount /mnt/cf             コンパクトフラッシュのアンマウント
#  exit                       root ユーザーから抜ける
```

7.5　カーネル起動パラメータ

それではコンパクトフラッシュをCATに装着し、電源を投入します。カーネルパラメータでルートパーテーション位置を指定します。ブートローダメニューから

```
>> setparam  console=ttySC0,115200 root=0301 ro
```

としてください。root=0301 は メジャー番号3（16進数）マイナー番号1（16進数）の意味で、/dev/hda1 を意味します。

7.6 Linuxの起動

ブートローダから boot とタイプすることで Linux が起動します。起動後、ネットワークデバイスドライバ・モジュールがロードされていないと思いますのでネットワークデバイスドライバ・モジュールをロードします。

```
# modprobe ne io=0x300
ne.c:v1.10 9/23/94 Donald Becker (becker@scyld.com)
Last modified Nov 1, 2000 by Paul Gortmaker
ASIX AX88796L ethercard probe at 0x300: 00 03 82 02 02 c4
eth%d: NE2000 found at 0x300, using IRQ 35.
```

次回からロード時に自動的に組み込まれるよう、/etc/modules に足します。

```
# vi /etc/modules
```

```
# at boot time, one per line. Lines beginning with "#" are ignored.
ne io=0x300
```

ネットワークの再起動

```
# /etc/init.d/networking restart
```

ディスク使用量の確認

この段階でのディスク使用量を確認します。119Mbyteのうち94Mバイト使用され、残り20Mバイトです。

```
# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda1             119M   94M   20M  84% /
tmpfs                  16M  4.0K   16M   1% /dev/shm
```

7.7 タイムゾーンの設定

タイムゾーンを Tokyo に設定します。

```
# tzconfig
Your current time zone is set to UTC
Do you want to change that? [n]: y              ← 変更するので Y

Please enter the number of the geographic area in which you live:

       5) Asia

Number: 5                                       ← 5番の Asia を選択する

Please enter the name of one of these cities or zones
You just need to type enough letters to resolve ambiguities
Press Enter to view all of them again

Name: [] Tokyo                                  ← Tokyo と入力する
```

デフォルトではシステム時計が UTC となっているので、設定を変更します。

```
#  vi /etc/default/rcS
```

```
UTC=no
```

この設定は再起動後から反映されます。

7.8 apt-get

Debian SH システムでは apt-get が使用できます。apt-get を使用して、インターネット上からアプリケーションパッケージの追加インストールが出来ます。

```
# apt-get update
# apt-get install gcc-3.4
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  binutils cpp-3.4
Suggested packages:
  binutils-doc gcc-3.4-doc
Recommended packages:
  libc6-dev
The following NEW packages will be installed:
  binutils cpp-3.4 gcc-3.4
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 5172kB of archives.
After unpacking 12.6MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

gcc をインストールすると CAT セルフでプログラムのコンパイルが出来るようになります。ここまで出来れば通常の PCと同じように使用することができます。

補足:NFSクライアントを使用するためには portmap が必須なので portmap のインストールをしておくと便利です。

```
# apt-get install portmap
```

8 最小ミニルートの構築

前章で作った Debian システムを小型化し、最小システムを作ります。オンボードフラッシュROM領域は7Mバイトなので7Mバイトに収まる小型システムを作ります。

8.1 ユーティリティーのインストール

miniroot は JFFS2 ファイル形式で保存します。JFFS2 形式のフォーマットを扱うユーティリティをインストールします。なお、通常はPCを用いてフォーマットを行いますが、CAT でセルフで実行することも出来ます。PCあるいは CAT で以下のコマンドを実行します。

```
# apt-get install mtd-tools
```

以上で mkfs.jffs2 コマンドがインストールされます。

8.2 依存ライブラリの調査

bash と必要コマンドのみの最小システムを作ります。bash を動作させるには bash が必要とするライブラリを調査します。以下のコマンドで bash が必要とするライブラリの一覧を取得することが出来ます。

```
# /lib/ld-linux.so.2 --list /bin/bash
        libncurses.so.5 => /lib/libncurses.so.5 (0x29559000)
        libdl.so.2 => /lib/tls/libdl.so.2 (0x295ac000)
        libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x295be000)
```

```
        libc.so.6 => /lib/tls/libc.so.6 (0x295da000)
        /lib/ld-linux.so.2 (0x52aaa000)
```

このように /bin/bash は /lib/libncurses.so.5 とリンクされていることが分かります。実際には /lib/libncurses.so.5 はシンボリックリンクになっている場合がありますので全てのファイルについてシンボリックリンク先も調べます。

```
# ls -l /lib/libncurses.so.5
lrwxrwxrwx  1 root root 17 Nov 14 11:20 /lib/libncurses.so.5 -> libncurses.so.5.4
```

この場合は /lib/libncurses.so.5.4 と /lib/libncurses.so.5 の双方のファイルが必要になります。

8.3 必要なファイルを集める

target というディレクトリを作り、必要なファイルを集めます。実行ファイル /bin/bash 、/bin/busybox 。そしてそのライブラリを cp -a でコピーします。

ディレクトリを作る

```
# mkdir target
# mkdir target/bin
# mkdir target/sbin
# mkdir target/lib
# mkdir target/lib/tls
# mkdir target/dev
# mkdir target/tmp
# chmod 1777 target/tmp
```

実行コマンド(bash, busybox)をコピーする

```
# cp -a /bin/bash               target/bin/
# cp -a /bin/busybox            target/bin/
```

必要なライブラリをコピーする

```
# cp -a /lib/libncurses.so.5    target/lib/
# cp -a /lib/libncurses.so.5.4  target/lib/
# cp -a /lib/tls/libdl.so.2     target/lib/tls/
# cp -a /lib/tls/libdl-2.3.5.so target/lib/tls/
# cp -a /lib/libgcc_s.so.1      target/lib/
# cp -a /lib/tls/libc.so.6      target/lib/tls/
# cp -a /lib/tls/libc-2.3.5.so  target/lib/tls/
# cp -a /lib/ld-linux.so.2      target/lib/
# cp -a /lib/ld-2.3.5.so        target/lib/
```

最低限のデバイスファイルをコピーする

```
# cp -a /dev/ttySC0  target/dev/
# cp -a /dev/console target/dev/
# cp -a /dev/null    target/dev/
```

busybox とはファイル操作などのコマンドを一つの実行ファイルにまとめたものです。busybox をシンボリックリンク名で呼び出すことで各種コマンドを実行することが出来ます。busybox へのシンボリックリンクを張ります。

```
# cd target/bin
# ln -s busybox ls
# ln -s busybox mount
# ln -s busybox ln
# ln -s busybox cp
# ln -s busybox mv
# ln -s busybox rm
```

```
# ln -s busybox mknod
# ln -s bash sh
# cd ../../
```

Linux カーネルは起動後 /sbin/init を最初に起動します。/sbin/init が /bin/bash となるよう、シンボリックリンクを張ります。

```
# cd target/sbin
# ln -s ../bin/bash init
# cd ../../
```

ここまでの作業で、target ディレクトリは以下のようになりました。

```
target/
|-- bin
|   |-- bash
|   |-- busybox
|   |-- cp -> busybox
|   |-- ln -> busybox
|   |-- ls -> busybox
|   |-- mknod -> busybox
|   |-- mount -> busybox
|   |-- mv -> busybox
|   |-- rm -> busybox
|   `-- sh -> bash
|-- dev
|   |-- console
|   |-- null
|   `-- ttySC0
|-- lib
|   |-- ld-2.3.5.so
|   |-- ld-linux.so.2 -> ld-2.3.5.so
|   |-- libgcc_s.so.1
|   |-- libncurses.so.5 -> libncurses.so.5.4
|   |-- libncurses.so.5.4
|   `-- tls
|       |-- libc-2.3.5.so
|       |-- libc.so.6 -> libc-2.3.5.so
|       |-- libdl-2.3.5.so
|       `-- libdl.so.2 -> libdl-2.3.5.so
|-- root
|-- sbin
|   `-- init -> ../bin/bash
`-- tmp
```

8.4 JFFS2 イメージを作る

次のコマンドで JFFS2 イメージファイルを作ります。

```
# mkfs.jffs2 -p -o rootfs.bin -r target/
    -p 残り部分を 0xFF で埋める  -o 出力ファイル  -r 元のディレクトリ
# ls -l
1376256 Nov 14 21:09 rootfs.bin
```

ファイルサイズ 1.376Mbyte の JFFS2 イメージが出来ました。

8.5 rootfs の書き込み、カーネル起動パラメータ

CAT のブートローダで rootfs.bin を ルートファイルシステム保存領域に書き込みます。

```
>> cp cf0:rootfs.bin rom:rootfs
```

また、ブートパラメータに root=1f02 rootfstype=jffs2 を指定してください。

```
>>setparam console=ttySC0,115200 root=1f02 rootfstype=jffs2 ro
console=ttySC0,115200 root=1f02 rootfstype=jffs2 ro
>boot
loading linux
Uncompressing Linux...
：略
init-3.00# ls
bin   dev   lib   sbin
```

以上の手順で最小構成の rootfs が構築できます。

8.6 自作ソフトの組み込み

4章で作成した LED のデバイスドライバ(cat709port.ko) と 5章で作成した LED を点滅させるアプリケーション (led_control)を最小 rootfs に組み込みます。前節の target の続きで実行します。

ディレクトリを作成します

```
# mkdir -p target/lib/modules
# mkdir -p target/usr/local/bin
```

CAT709 から ismod コマンドをコピーします。また、前章で作成した cat709port.ko ドライバと led_control プログラムをそれぞれコピーします。

```
# cp -a /CAT709 のディレクトリ/sbin/insmod target/sbin
# cp -a /cdrom/sample-driver/cat709port/cat709port.ko target/lib/modules/
# cp -a /cdrom/sample-driver/cat709port/led_control   target/usr/local/bin/
```

LED0 のデバイスファイルを作ります

```
# mknod target/dev/led0 c 241 4
```

/sbin/init はシェルスクリプトとします。

```
# rm target/sbin/init
# vi target/sbin/init
```

/sbin/init スクリプトの中身

```
#!/bin/sh

# load module
/sbin/insmod /lib/modules/cat709port.ko

# apprication program
/usr/local/bin/led_control &

while true
do
        echo okay.....
        sleep 10
done
```

スクリプトに実行属性を付けます。

```
# chmod +x target/sbin/init
```

最後に busybox を使って sleep コマンドと echo コマンドを作ります。

```
# cd target/bin/
# ln -s busybox sleep
# ln -s busybox echo
# cd ../../
```

ここまでの作業で target は次のようになりました。

```
target/
|-- bin
|   |-- bash
|   |-- busybox
|   |-- cp -> busybox
|   |-- echo -> busybox
|   |-- ln -> busybox
|   |-- ls -> busybox
|   |-- mknod -> busybox
|   |-- mount -> busybox
|   |-- mv -> busybox
|   |-- rm -> busybox
|   |-- sh -> bash
|   `-- sleep -> busybox
|-- dev
|   |-- console
|   |-- led0
|   |-- null
|   `-- ttySC0
|-- lib
|   |-- ld-2.3.5.so
|   |-- ld-linux.so.2 -> ld-2.3.5.so
|   |-- libgcc_s.so.1
|   |-- libncurses.so.5 -> libncurses.so.5.4
|   |-- libncurses.so.5.4
|   |-- modules
|   |   `-- cat709port.ko
|   `-- tls
|       |-- libc-2.3.5.so
|       |-- libc.so.6 -> libc-2.3.5.so
|       |-- libdl-2.3.5.so
|       `-- libdl.so.2 -> libdl-2.3.5.so
|-- root
|-- sbin
|   |-- init
|   `-- insmod
|-- tmp
`-- usr
    `-- local
        `-- bin
            `-- led_control
```

JFFS2 ファイルイメージを作ります。

```
# mkfs.jffs2 -p -o rootfs.img -r target/
# ls -l rootfs.bin
-rw-r--r--  1 root root 1376256 Nov 14 22:18 rootfs.bin
```

先ほどと同様に ブートローダメニューで rootfs.bin ファイルを書き込み、起動します。

```
>> cp cf0:rootfs.bin rom:rootfs
>> boot
loading linux
Uncompressing Linux...
：略
cat709port hello
okay.....
点灯
消灯
点灯
消灯
点灯
消灯
```

LED が点滅していることを確認します。

9 その他の機能

9.1 SRAM

9.2 CompactFlash

シリコンリナックス株式会社

2005. 11月 初版

2005. 12月5日 誤記修正

2005. 12月22日 新 bootloader 対応書き直し