SUZAKU
スターターキットガイド
（Linux 開発編）

Version 1.1.0

2006 年 10 月 20 日

株式会社アットマークテクノ
http://www.atmark-techno.com/

SUZAKU 公式サイト
http://suzaku.atmark-techno.com/

はじめに

SUZAKU スターターキットは、SUZAKU を初めて手に取る方にもお使いいただけるように、必要な機材をセットにした SUZAKU 学習用キットです。

SUZAKU（朱雀）は、FPGA を搭載した組み込み向け小型汎用ボードです。まず、SUZAKU の象徴である FPGA について簡単に説明します。FPGA（Field Programmable Gate Array）とは、プログラミングすることができる LSI の 1 つで、プロセッサや設計図を送り込んでシミュレーションを繰り返しできるのが特徴です。この特徴を生かし、ASIC の動作確認用の試作や、仕様変更が見込まれる製品などに用いられています。

SUZAKU は、このような FPGA の利点を生かした次世代プラットフォームとして開発されました。SUZAKU には、以下のような特長があります。

- FPGA
  Xilinx 社の最新 FPGA を採用し、大規模で柔軟な拡張をすることができます。SUZAKU-S では低コストな Spartan-3E, Spartan-3 を、SUZAKU-V では高性能な Virtex-II Pro を採用しています。

- Function
  Xilinx 社またはサードパーティ各社から供給される豊富な IP（Intellectual Property）コアを利用することで、必要な機能を容易に追加することできます。

- CPU
  SUZAKU-S では低コストで資産継承性が高いソフトプロセッサ「MicroBlaze」を、SUZAKU-V では高性能で実績の高いハードプロセッサ「PowerPC405」を採用しています。

- I/O
  小型ボードサイズながら豊富な I/O ピンを持ち、自由に拡張することができます。

- Linux
  各種ネットワークのプロトコルスタックからファイルシステムまで安定した実績のある OS 環境を提供します。SUZAKU-S では MMU 不要の uClinux を、SUZAKU-V では標準的な Linux を採用しています。

- Software
  デバイスドライバから各種サーバソフトウェアまで、オープンソースで開発された Linux 対応の豊富なソフトウェア資産を活用することができます。実績のある安定したソフトウェアは開発期間を短縮します。

- Network
  ボードに標準搭載されている LAN インターフェース（10BASE-T/100BASE-TX）と Linux の提供する TCP/IP プロトコルスタックを組み合わせ、容易にネットワーク対応機器の開発を実現します。

以上のことから、SUZAKU をベースにシステム（とりわけネットワーク対応機器）を開発する際に、開発期間の短縮やコストダウンを図ることができます。

SUZAKU スターターキットは、SUZAKU を使った機器開発の体験・修得をお手伝いするものです。SUZAKU の開発は、大きく「FPGA」と「ソフトウェア」の 2 つに分けて考えることができます。そこで、『SUZAKU スターターキットガイド（FPGA 開発編）』と『SUZAKU スターターキットガイド（Linux 開発編）』の 2 つのガイドを用意しました。各ガイドには読み始める順番はありませんので、興味のある開発編から取り組むことができます。

SUZAKU スターターキットを足掛かりに開発手法を修得していただき、SUZAKU の可能性を存分に引き出していただければ幸いです。

対象となる読者

本書は、SUZAKU の組み込みソフトウェア開発者向けに書かれた入門書です。ここで言うソフトウェアは、OS 上で実行するユーザアプリケーションとデバイスドライバを指しています。SUZAKU では、OS に Linux (または uClinux)を採用していますので、世の中に存在する多数の Linux 関連情報を参考に開発していただけます。

しかし、組み込みシステム開発で用いられている「クロス開発」と呼ばれる手法や、SUZAKU で採用しているディストリビューション（uClinux-dist）を利用した開発は、情報も少なく初めての方には分かりにくく感じられるのではないでしょうか。そのため、SUZAKU で実際に開発しようと思っても、「何をすればよいか分からないです。」という人もおられると思います。

本書では、このような製品付属のソフトウェアマニュアルでは難しかった方や、SUZAKU を初めて使う方など、開発方法について丁寧で分かりやすい説明を望む開発者を対象にしています。

本書の構成

本書では、SUZAKU を手にしてから開発を行なうまでの手順に従い説明します。内容は２部構成となっています。

まず前半では、SUZAKU を手にしてから動かすために必要な準備作業（第１章）と、基本的な操作方法（第２章）について説明します。この前半で説明する内容は、SUZAKU 全ボードに共通するものです。SUZAKU スターターキットに限定された話ではありませんので、既に SUZAKU をお使いの読者は、読み飛ばしていただいても構いません。

後半では、ソフトウェア開発を体験していただきます。ここでいうソフトウェア開発とは、SUZAKU に標準インストールされている Linux で動作するアプリケーション開発と、Linux 用のデバイスドライバ開発のことです。最初に、開発準備として作業用 PC にクロス開発環境を構築します（第３章）。次に、実際に小規模な CGI アプリケーションを作成し、アプリケーション開発の基本事項を学びます（第４章）。その後、仮想のデバイスドライバを作って、デバイスドライバの基礎とアプリケーションからの利用方法を体験します（第５章）。最後に、SUZAKU スターターキットの LED/SW ボードを操作するソフトウェア開発を紹介します（第６章）。

必要となる知識

本書は、読者が UNIX に関する基本的な知識をすでにお持ちで、`ls` や `cd` など基本的なコマンドを操作できること、さらに C 言語・Makefile によるプログラミングの経験を多少なりともあることを前提としています。なお、プログラミングの経験は、UNIX でのプログラミングである必要はなく、MS-DOS または Microsoft Windows（以降、Windows と略記）でのプログラミングであってもかまいません。

表記について

このマニュアルでは以下のようにフォントを使っています。

表 0-1 使用しているフォント

フォント例	説明
本文中のフォント	本文
[PC ~]$ ls	プロンプトとユーザ入力文字列

また、このマニュアルに記載されているコマンドの入力例は、表示されているプロンプトによって、それぞれに対応した実行環境を想定して書かれています。「/」の部分はカレントディレクトリによって異なります。各ユーザのホームディレクトリは「~」で表わします。

表 0-2 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
[PC /]#	作業用 PC 上の特権ユーザで実行
[PC /]$	作業用 PC 上の一般ユーザで実行
[SUZAKU /]#	SUZAKU 上の特権ユーザで実行

謝辞

SUZKAU で使用しているソフトウェアは Free Software / Open Source Software で構成されています。Free Software / Open Source Software は世界中の多くの開発者の成果によって成り立っています。この場を借りて感謝の意を示します。

ソフトウェアに関する注意事項

本製品に含まれるソフトウェア（付属のドキュメント等も含みます）は、現状のまま（AS IS）提供されるものであり、特定の目的に適合することや、その信頼性、正確性を保証するものではありません。また、本製品の使用による結果についてもなんら保証するものではありません。

保証に関する注意事項

製品保証範囲について

付属品（ソフトウェアを含みます）を使用し、取扱説明書、各注意事項に基づく正常なご使用に限り有効です。万一正常なご使用のもと、製品が故障した場合は故障箇所の修理をさせていただきます。

保証対象外になる場合

次のような場合の故障・損傷は、保証期間内であっても保証対象外になります。

1. 取扱説明書に記載されている使用方法、または注意に反したお取り扱いによる場合
2. 改造や部品交換に起因する場合。または正規のものではない機器を接続したことによる場合
3. お客様のお手元に届いた後の輸送、移動時の落下など、お取り扱いの不備による場合
4. 火災、地震、水害、落雷、その他の天災、公害や異常電圧による場合
5. AC アダプタ、専用ケーブルなどの付属品について、同梱のものを使用していない場合
6. 修理依頼の際に購入時の付属品がすべて揃っていない場合

免責事項

弊社に故意または重大な過失があった場合を除き、製品の使用および、故障、修理によって発生するいかなる損害についても、弊社は一切の責任を負わないものとします。

本製品は購入時の初期不良以外の保証をおこなっておりません。保証期間は商品到着後２週間です。本製品をご購入されましたらお手数でも必ず動作確認をおこなってからご使用ください。本製品に対して注意事項を守らずに発生した故障につきましては保証対象外となります。

商標について

記載の会社名、製品名はそれぞれの登録商標または商標です。

ダウンロード

本書で紹介するソースコードやファイルは、機能増強や不具合解決等のアップグレードを行うことがあります。下記サイトに最新版がございますので、ダウンロードしてお使いください。

開発に関するファイル
http://suzaku.atmark-techno.com/downloads/all

各種ドキュメント
http://suzaku.atmark-techno.com/downloads/docs

目次

表目次

図目次

1. 作業の前に

SUZAKU スターターキットに電源を入れる前に、必ず行う作業について説明します。
最初に必要な物の確認を行ない、次に最低限必要なソフトウェアの準備を行ないます。最後に SUZAKU スターターキットの各部名称について説明します。

1.1. ハードウェアの準備

1.1.1. SUZAKU スターターキットの内容物確認

まず、はじめに SUZAKU スターターキットの内容物を確認してください。

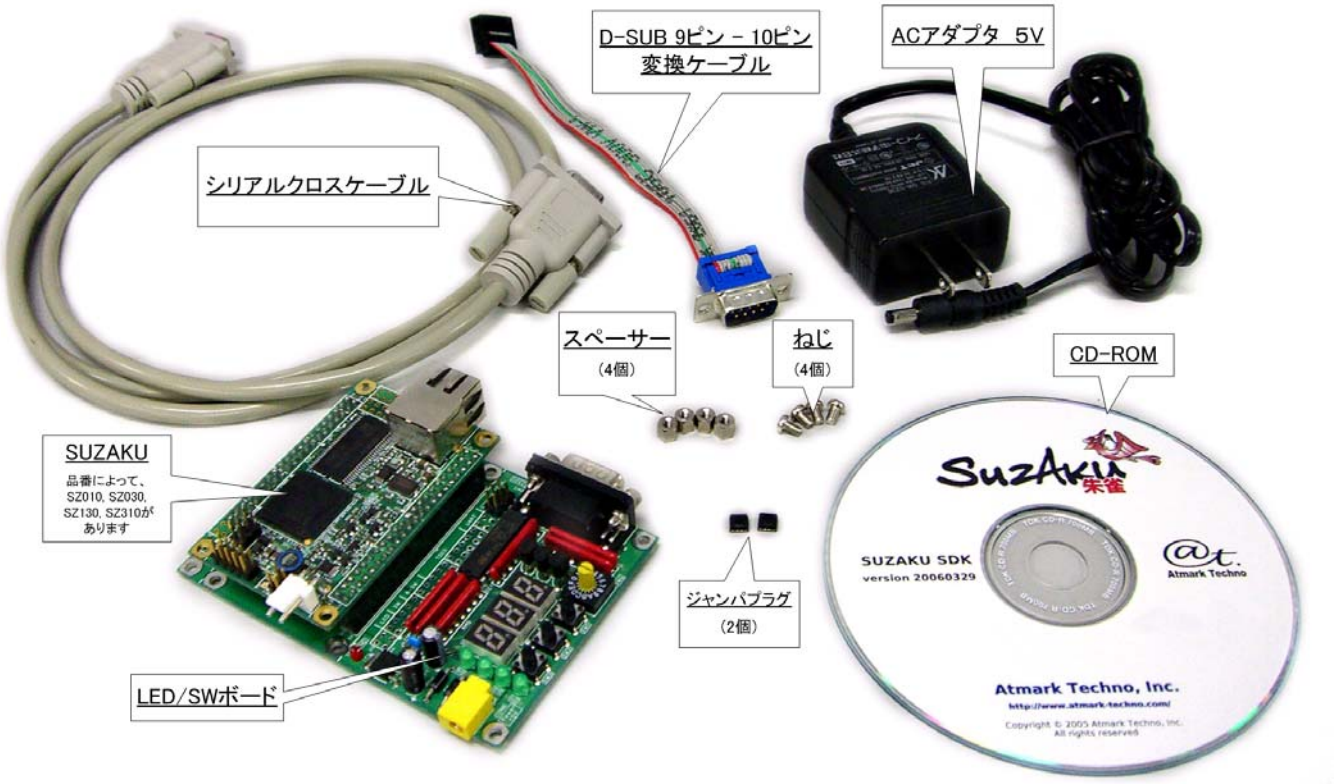

図 1-1 SUZAKU スターターキット内容物

1.1.2. 作業用 PC の準備

本書にそって SUZAKU スターターキットをご使用になる場合は、以下の条件を満す PC を一台準備してください。

OS	Windows XPもしくはLinux[1]が動作すること 管理者(Administrator、root)権限で操作できること
CPU	1GHz クラス以上のもの推奨
メモリ	256MB 以上 (WindowsXP の場合、512MB 以上推奨) 起動時に 64MB 以上のメモリを利用するアプリケーションが利用可能であること
シリアル通信ポート	シリアル通信ポートが一つ使用可能であること または USB-シリアルケーブルによるシリアル接続が可能であること
ネットワーク	有線 LAN ポート (10/100Base-T) によるイーサネット接続が可能であること
ハードディスクドライブ	特定のドライブに 4GB 程度以上の空き容量があること

1.1.3. 接続

内容物をすべて確認し、作業用PCの準備が整いましたら図 1-2を参考に、シリアルクロスケーブル、電源、そしてLANケーブルをSUZAKUに接続してください。

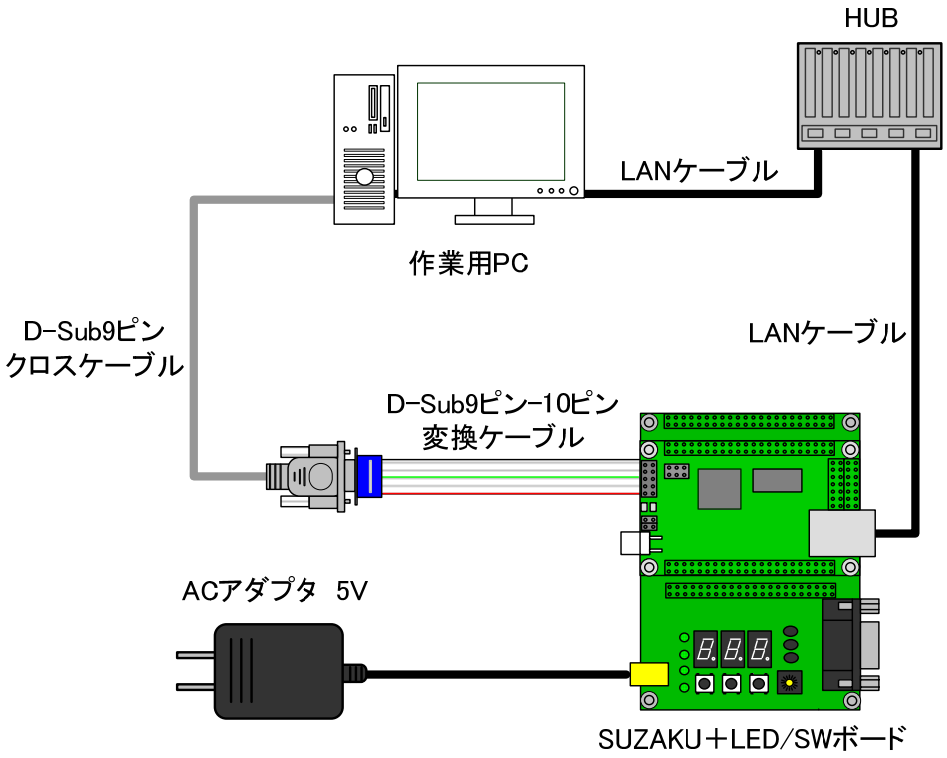

図 1-2 SUZAKU スターターキット接続図

[1] 本書では、Debian GNU Linuxを使用します。

1.2. ソフトウェアの準備

SUZAKU は、シリアルポートをコンソールとして使用します。SUZAKU のコンソールから出力される情報を読み取ったり、SUZAKU のコンソールに情報を送ったりするには、シリアル通信用のソフトウェアが必要です。

下記手順に従って、シリアル通信ソフトウェアをインストールしてください。なお、本書では、Windows 用として Tera Term Pro を、Linux 用として minicom を利用することとします。この他にもシリアル通信ソフトウェアは存在します。すでに使い慣れたソフトウェアをお持ちの方は、そちらをお使いいただいても構いません。

1.2.1. シリアル通信ソフトウェアのインストール

1) Windows の場合

Tera Term Pro を作業用 PC にインストールします。この手順は、管理者（Administrator）権限を持つユーザで行ってください。

① Tera Term Pro インストーラの実行

setup.exe を実行します。

② 言語の選択

Language リストから「Japanese」を選択して、「Continue」ボタンを押下します。

③ 確認ダイアログ

確認ダイアログが表示されますので、「Continue」ボタンを押下します。

④ キーボードの選択

キーボードを選択します。通常は、「IBM PC/AT (DOS/V)キーボード」を選択します。選択が完了したら、「Continue」ボタンを押下します。

⑤ インストール先の指定

インストール先を指定するダイアログが表示されます。C ドライブに十分な空き容量がある場合、デフォルトのままで構いません。他のドライブにインストールする場合などは、任意のフォルダを指定してください。フォルダ指定が完了したら、「Continue」ボタンを押下します。すぐにインストールが開始されます。

⑥ インストール完了

数秒程度でインストールが完了し、Setup 完了ダイアログが表示されます。「OK」を押してインストール作業を完了します。

2) Linux の場合

minicom をインストールします。必ず root 権限で行なってください。minicom のパッケージファイルは付属 CD に用意されています。

```
[PC ~]# dpkg -i minicom_2.1-4.woody.1_i386.deb
```

図 1-3 minicom のインストール

1.2.2. シリアル通信ソフトウェアのアンインストール

1) Windows の場合
開発作業終了後、Tera Term Pro をアンインストールする場合、Windows のコントローラパネルにある「プログラムの追加と削除」からアンインストールしてください。

2) Linux の場合
開発作業終了後、minicom をアンインストールする場合、minicom のパッケージを削除してください。

```
[PC ~/]# dpkg -r minicom
```

図 1-4 minicom のアンインストール

1.2.3. シリアル通信ソフトウェアの設定

SUZAKU のシリアルポート 1(CON1)と作業用 PC をシリアルケーブルで接続し、シリアル通信ソフトウェアを起動します。次のように通信設定を行なってください。

表 1-1 シリアル通信設定

項目	設定
転送レート	115,200bps
データ長	8bit
ストップビット	1bit
パリティ	なし
フロー制御	なし

1) Windows の場合
Tera Term Pro を起動し、Setup メニューから「Serial port...」を選択し、通信設定を行います。

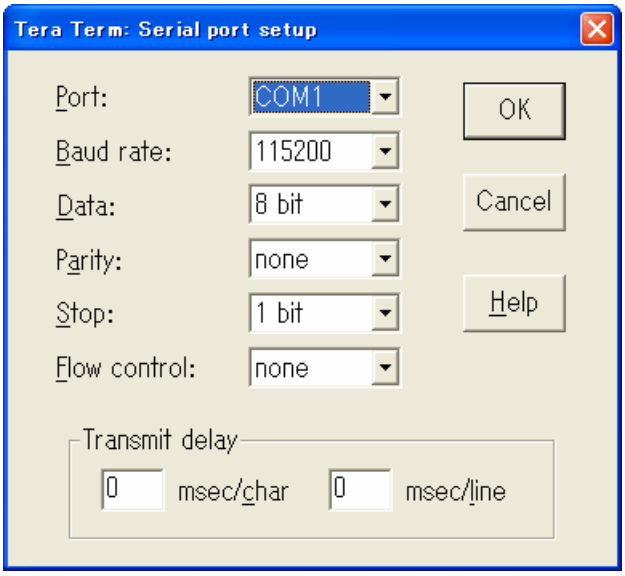

図 1-5 シリアル通信設定 (Tera Term Pro)

2) Linux の場合

`minicom` に`-s` オプションを付けて起動します。`-s` を指定することで、設定画面に移行します。シリアルポートの通信設定を行ってください。

```
[PC ~/]$ minicom -s
```

図 1-6 minicom の起動 (-s)

図 1-7 minicom の起動画面 (-s)

カーソルを「シリアルポート」にあわせ、「Enter」キーを押下します。

図 1-8 シリアル通信設定 (minicom)

「シリアルデバイス」、「速度/パリティ/ビット」、「ハードウェア流れ制御」および「ソフトウェア流れ制御」を表 1-1のように設定してください。設定する際は、各項目の左端にあるアルファベットをキー入力して行います。設定の変更が完了したら、「Esc」キーを入力しメイン設定画面に戻ります。

設定内容をデフォルトとして保存しておくと次回以降この設定作業を省略することができます。「”dfl”に設定を保存」にカーソルをあわせ、「Enter」キーを押下します。

図 1-9 シリアル通信設定の保存 (minicom)

「終了」を選択し、「Enter」キーを押します。

TIPS

minicom の初期設定では、起動時にモデムの初期化を行うようになっていることが多いようです。設定で初期化用の AT コマンドを外すか、minicom に-o オプションを付けて起動することでモデム初期化コマンドを省略することができます。

また、使用するシリアルポートの読み込みと書き込み権限が無い場合、minicom の起動に失敗します。使用するシリアルポートの権限を確認してください。詳しくは minicom のマニュアルまたはご使用の OS のマニュアルをご覧ください。

1.3. 各部の名称

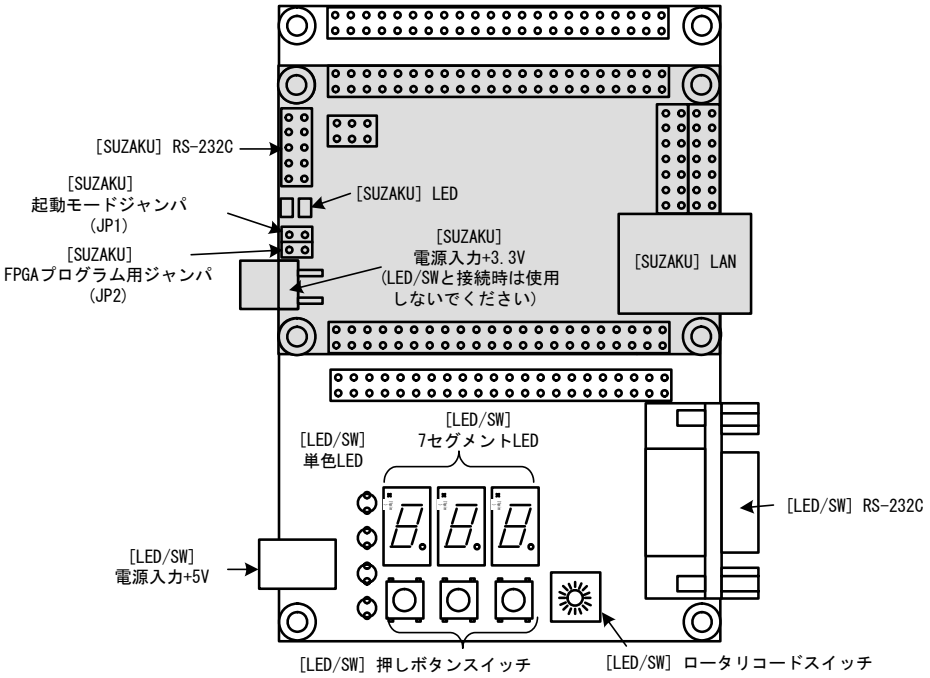

図 1-10 各種インターフェースの配置

1.3.1. ジャンパ

SUZAKU ではジャンパの設定を変えることで、起動時の動作を変更することができます。以下の表に設定と動作の関連を記載します。

表 1-2 ジャンパの設定と起動時の動作

JP1	JP2	起動時の動作	起動モード
オープン	オープン	Linux カーネルを起動	オートブートモード
ショート	オープン	ファーストステージブートローダーを起動	ブートローダーモード
－	ショート	FPGAコンフィギュレーション[2]	－

TIPS

ジャンパのオープン、ショートとは

・オープン　：　ジャンパピンにジャンパソケットを挿さない状態
・ショート　：　ジャンパピンにジャンパソケットを挿した状態

[2]詳しくは、ハードウェアマニュアルを参照してください

2. 電源を入れてみよう

この章では、SUZAKU の電源を投入し実際に動かしてみます。SUZAKU の起動や終了など基本的な使用方法について説明します。

2.1. 起動

起動モードジャンパ（JP1）と FPGA プログラム用ジャンパ（JP2）がオープンになっていることを確認して、電源を入れてください。SUZAKU スターターキットには電源ボタンのようなものはありませんので、AC アダプタを差し込むことで電源が入ります。

SUZAKUが正常に起動した場合、シリアル通信ソフトウェアに図 2-1のようなログが出力されます。これは Linux の起動ログです（SUZAKU-S スターターキット，ディストリビューション：uClinux-dist-20051110-suzaku4）。

```
Linux version 2.4.32-uc0 (atmark@atde) (gcc version 3.4.1 ( Xilinx EDK 8.1 Build EDK_I.17 090206 ))
#1 Mon Oct 16 19:22:07 JST 2006
On node 0 totalpages: 8192
zone(0): 8192 pages.
zone(1): 0 pages.
zone(2): 0 pages.
CPU: MICROBLAZE
Kernel command line:
Console: xmbserial on UARTLite
Calibrating delay loop... 25.44 BogoMIPS
Memory: 32MB = 32MB total
Memory: 29484KB available (986K code, 1933K data, 44K init)
Dentry cache hash table entries: 4096 (order: 3, 32768 bytes)
Inode cache hash table entries: 2048 (order: 2, 16384 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 8192 (order: 3, 32768 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Microblaze UARTlite serial driver version 1.00
ttyS0 at 0xffff2000 (irq = 1) is a Microblaze UARTlite
Starting kswapd
xgpio #0 at 0xFFFFA000 mapped to 0xFFFFA000
Xilinx GPIO registered
sil7segc (1.0.1): 7seg-LED Driver of SUZAKU I/O Board -LED/SW- for CGI demo.
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
eth0: LAN9115 (rev 1150001) at ffe00000 IRQ 2
Suzaku MTD mappings:
  Flash 0x800000 at 0xff000000
flash: Found an alies 0x800000 for the chip at 0x0, ST M25P64 device detect.
Creating 7 MTD partitions on "flash":
0x00000000-0x00800000 : "Flash/All"
0x00000000-0x00100000 : "Flash/FPGA"
0x00100000-0x00120000 : "Flash/Bootloader"
0x007f0000-0x00800000 : "Flash/Config"
0x00120000-0x007f0000 : "Flash/Image"
0x00120000-0x00420000 : "Flash/Kernel"
0x00420000-0x007f0000 : "Flash/User"
FLASH partition type: spi
uclinux[mtd]: RAM probe address=0x8012ba4c size=0x1af000
uclinux[mtd]: root filesystem index=7
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 2048 bind 4096)
VFS: Mounted root (romfs filesystem) readonly.
Freeing init memory: 44K
Mounting proc:
Mounting var:
Populating /var:
Running local start scripts.
Mounting /etc/config:
Populating /etc/config:
flatfsd: Created 4 configuration files (149 bytes)
Setting hostname:
Setting up interface lo:
Starting DHCP client:
Starting inetd:
```

```
Starting thttpd:

SUZAKU-S.STARTER-KIT login:
```

図 2-1 起動ログ

2.2. ログイン

起動が終了すると、シリアル通信ソフトウェアにログインプロンプトが表示されます。初期設定の SUZAKU スターターキットでは、root のアカウントだけが準備されています。以下の図表を参考にログインしてください。

表 2-1 コンソールログイン時のユーザ名とパスワード

ユーザ名	パスワード	権限
root	root	特権ユーザ

ログインに成功するとプロンプト(#)が表示され、コマンド入力待ち状態になります。

```
SUZAKU-S.STARTER-KIT login: root
Password:                                   ←パスワードは表示されません


BusyBox v1.00 (2006.09.24-10:36+0000) Build-in shell (msh)
Enter ‘help’ for a list of built-in commands.

#
```

図 2-2 ログイン

2.3. ログアウト

SUZAKU スターターキットからログアウトするには exit コマンドを使います。exit コマンドを入力すると再びログインプロンプトが表示されます。

```
[SUZAKU /]# exit

SUZAKU-S.STARTER-KIT login:
```

図 2-3 ログアウト

2.4. 終了

SUZAKU スターターキットには電源ボタンがありません。終了するには、電源を切断する必要があります。AC アダプタをコンセントから抜いて終了してください。

2.5. ネットワーク設定

初期状態のSUZAKUスターターキットは、DHCPを使用してIPアドレスを取得する設定になっています。ネットワーク設定を変更する場合は、イメージを再作成する必要があります。「3.3. . はじめてのコンパイルとSUZAKUへのインストール」および参考文献[2]もあわせて参照してください。

2.5.1. ネットワーク設定の確認

ネットワークの設定は `ifconfig` コマンドで確認することができます。詳しくは、`ifconfig` コマンドのマニュアルを参照してください。DHCP にて IP アドレスの取得に成功した場合は、以下のように結果が表示されます。赤枠の `inet addr` に続く数字列が IP アドレスを示しています。

```
[SUZAKU /]# ifconfig
eth0      Link encap:Ethernet  HWaddr XX:XX:XX:XX:XX:XX
          inet addr:192.168.1.xx  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
                 Interrupt:2 Base address:0x300

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

図 2-4 ifconfig 実行例

2.5.2. 固定 IP アドレスで使用する場合

ご利用の環境にDHCPサーバが存在しない場合は、起動時にIPアドレスの取得に失敗します。その際には、固定IPアドレスを設定してください。固定IPアドレスは、`ifconfig`コマンドで設定することができます。図 2-5に、IPアドレスに 192.168.1.100 を設定する例を示します。設定終了後、`ifconfig`コマンドで正しく設定されているかを確認してください。

```
[SUZAKU /]# ifconfig eth0 192.168.1.100
```

図 2-5 IP アドレスの設定

なお、この方法による設定は、再起動すると無効となります。電源投入時から固定IPアドレスを使用するには、「3.3. はじめてのコンパイルとSUZAKUへのインストール」で説明するフラッシュイメージの変更が必要です。詳しくは、参考文献[1]を参照してください。

2.6. telnet ログイン

次のユーザ名／パスワードで telnet ログインが可能です。

表 2-2 telnet ログイン時のユーザ名とパスワード

ユーザ名	パスワード
root	root

2.7. ファイル転送

ftpによるファイル転送が可能です。表 2-3に示すユーザとパスワードでログインしてください。ホームディレクトリは「/」です。書き込みは、「/var/tmp」ディレクトリ以下だけ許可されています。ファイル転送（putコマンド）を実行する場合にはディレクトリを移動してから行なってください。

表 2-3 ftp のユーザ名とパスワード

ユーザ名	パスワード
root	root

2.8. Web サーバ

thttpdという小さなHTTPサーバが起動しており、WebブラウザからSUZAKUをブラウズすることが出来ます。データディレクトリは「/home/httpd」です。URLは「http://(SUZAKUのIPアドレス)/」になります（例 http://192.168.1.100/）。

図 2-6 Web サーバ

3. 開発環境の構築

この章では、以降の章で必要となる開発環境を整えます。すでに SUZAKU の開発環境をお持ちの方はこの章を読み飛ばしても問題ありません。

最初に SUZAKU のソフトウェア開発に必須の Linux 環境を構築します。Linux 環境用にパソコンを用意できない方のために、Windows 上で仮想的に Linux 環境を構築することができる coLinux を紹介します。次に必要なソフトウェアをインストールしていきます。コンパイラやライブラリ、さらにコンパイルしたアプリケーションやカーネルなどを SUZAKU に書き込むためのツールもインストールします。必要なソフトウェアをすべてインストールした後は、SUZAKU 上で動くソフトウェアをソースコードからコンパイルしてみます。コンパイルできたソフトウェアを SUZAKU に書き込んで同じく動作するところまで確認します。

3.1. Windows 上に Linux 環境を構築する

SUZAKUの開発ツールは、Linux用として存在します。そのため、Windowsをお使いの方は、Windows上にLinux環境を構築する必要があります。本書では、coLinux（http://www.colinux.org/）を利用して、Windows上にLinux環境を構築する方法を説明します。

3.1.1. coLinux のインストール

1) 付属 CD の colinux フォルダにある coLinux-0.6.4.exe を実行します。
2) インストール先フォルダには c:¥coLinux を指定し、それ以外はデフォルトの設定のままでインストール作業を行ないます。

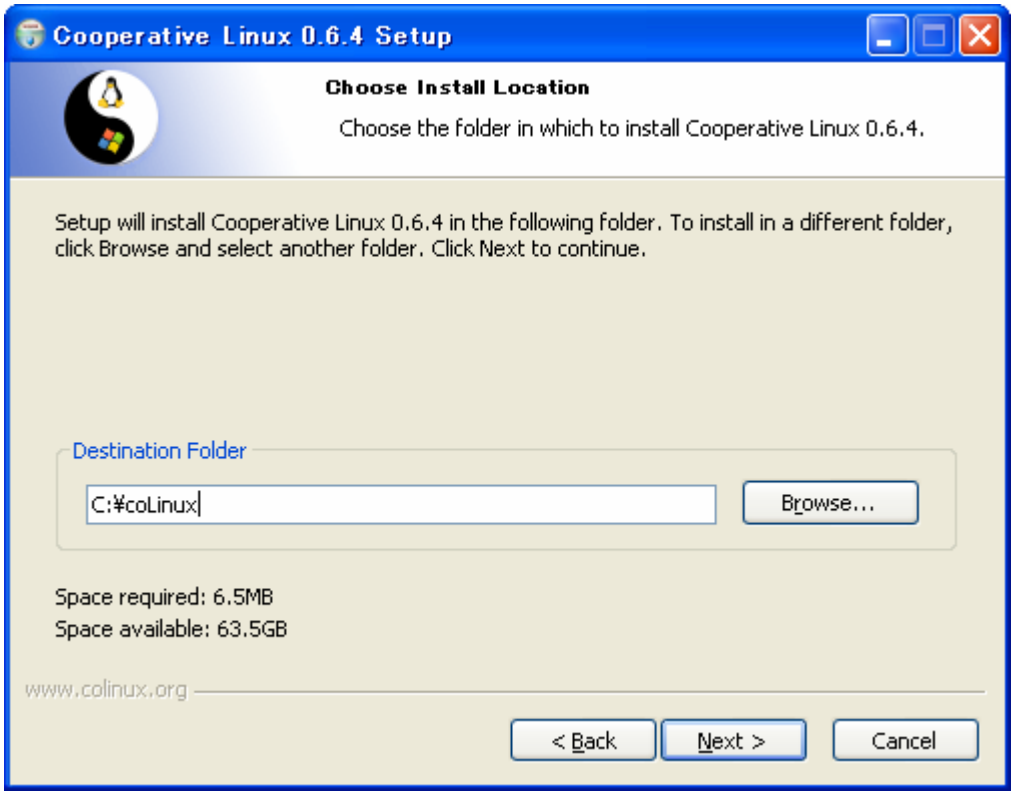

図 3-1 インストール先フォルダの指定

TIPS
インストール先に他のフォルダを指定する場合は、次の手順で用意する default.colinux.xml を編集し、フォルダ名を適切に変更する必要があります。

3.1.2. 環境構築用ファイルの準備

付属 CD の colinux フォルダから以下のファイルを用意し、coLinux のインストールフォルダ(C:¥coLinux)に展開します。

- root_fs.zip (ルートファイルシステム)
- swap_device_256M.zip (swap ファイルシステム)
- home_fs_2G.zip (/home にマウントされるファイルシステム)
- default.colinux.xml.zip (デバイス情報の設定ファイル)

TIPS
swap_device_..., home_fs_... のファイル名の数値は展開後のファイルサイズです。他のサイズのファイルも用意していますので、必要と思われるサイズのファイルを展開してください。展開ソフトによっては展開に失敗する場合があります。WindowsXP の標準機能で正常に展開できることを確認してあります。

3.1.3. coLinux の実行

1) DOS プロンプトを起動し、インストールフォルダ(C:¥coLinux)に移動します。
2) 「colinux-daemon.exe -c default.colinux.xml」とコマンド入力します。
3) 起動ログの表示後「colinux login:」と表示されたら「root」ユーザでログインします。

TIPS
colinux の実行時に、青画面(DRIVER_IRQL_NOT_LESS_OR_EQUAL)になった時は、C:¥boot.ini の"/noexecute=option"を"/noexecute=AlwaysOff"と書き換えてみてください。

3.1.4. 仮想ネットワークデバイスの設定

coLinux 側のネットワークに接続される仮想ネットワークデバイスについて、ネットワークの設定を行います。

1) 「スタートメニュー」-「マイ ネットワーク」-「ネットワーク接続を表示する」を実行し、ネットワーク接続アイコンのプロパティを開きます。

図 3-2 ネットワークプロパティの表示

2) [詳細設定タブ]をクリックし、インターネット接続の共有にある接続許可のチェックボックスにチェックをつけて、[OK ボタン]を押します。

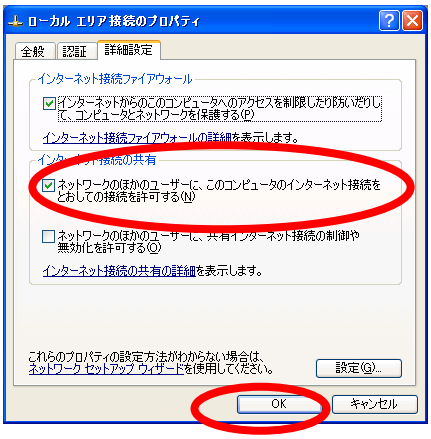

図 3-3 インターネット接続共有

3) 1)と同様に、仮想ネットワークデバイス（TAP）の「プロパティ」を開きます。

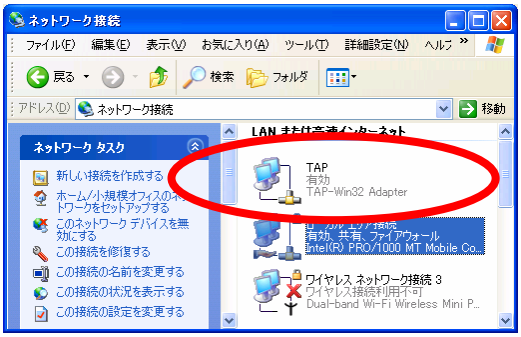

図 3-4 TAP のプロパティの表示

4) 「全般」タブにある「インターネット プロトコル（TCP/IP）」の「プロパティ」を開き、以下を設定します（デフォルトで以下の状態の場合、変更の必要はありません）。
- 「次の IP アドレスを使う」
- 「IP アドレス」を「192.168.0.1」
- 「サブネット マスク」を「255.255.255.0」

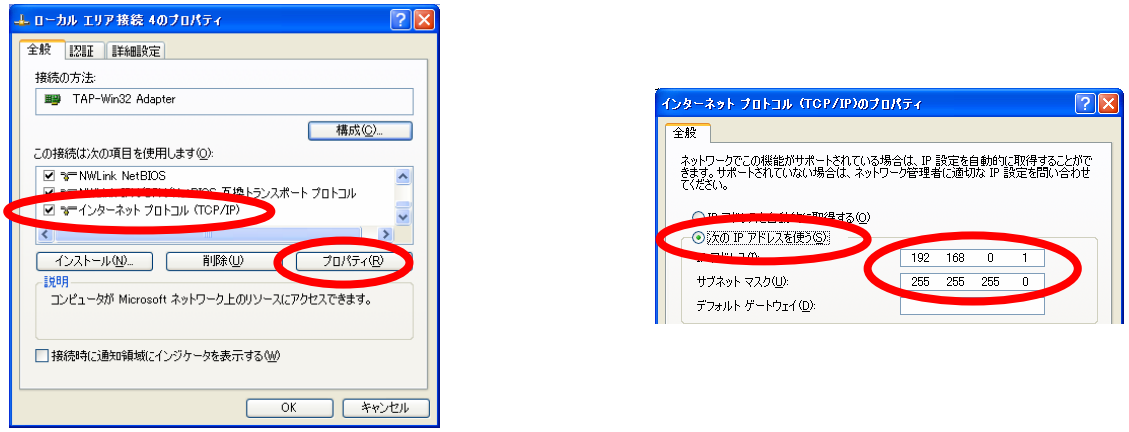

図 3-5 TAP のプロパティと IP アドレスの確認

5) 「OK」ボタンを押し、ネットワーク設定を完了します。

3.1.5. coLinux のネットワーク設定

coLinux では Windows とは別の IP アドレスを持ち、Windows を介してネットワークにアクセスするため、Windows のネットワーク設定の変更が必要となります。
設定方法には「ルーター接続」「ブリッジ接続」などがありますが、ここでは「ルーター接続」の方法を説明します。

1) コントロールパネルから「ネットワーク接続」を開きます。
2) 外部に接続しているネットワークを右クリックして「プロパティ」を開きます。
3) 「詳細設定」タブを開き、インターネット接続の共有を有効にします。

次に、ネットワークの設定を有効にするためのコマンドを coLinux 上で実行します。

```
colinux:~# /etc/init.d/networking restart
Reconfiguring network interfaces: done.
colinux:~#
```

図 3-6 ネットワークの設定コマンド

「ルーター接続」では 192.168.0.0/24 のネットワークアドレスが自動的に使用されるため、外部接続用のネットワークアドレスが同じ 192.168.0.0/24 の場合、設定に失敗します。この場合は外部接続用のネットワークアドレスを変更してください。
外部接続用のネットワークアドレスを変更できない場合「3.1.8. 特殊な場合の Windowsネットワーク設定方法」を参照してください。

3.1.6. coLinux ユーザの作成

coLinux の画面で以下のようにコマンドを入力し作業用ユーザを作成します。適宜パスワードなどを設定してください。

```
colinux:~# adduser somebody
Adding user somebody...
Adding new group somebody (1000).
Adding new user somebody (1000) with group somebody.
Creating home directory /home/somebody.
Copying files from /etc/skel
Enter new UNIX password:
```

図 3-7 ユーザ「somebody」を作業用ユーザとして追加する場合

3.1.7. Windows-coLinux 間のファイル共有

Windowsの共有フォルダを利用して、coLinuxとWindows間でファイルを交換する方法です。coLinuxの画面で図 3-8のようにsmbmountコマンドを実行して、共有フォルダのパスワードを入力してください。

```
colinux:~# mkdir /mnt/smb
colinux:~# smbmount //192.168.0.100/shared /mnt/smb
212: session request to 192.168.0.100 failed (Called name not present)
212: session request to 192 failed (Called name not present)
Password:
```

図 3-8 Windows の IP アドレス:192.168.0.100、共有フォルダ名：shared の場合

ユーザ名が Windows 側と異なる場合は、ユーザ名をコマンドのオプションで指定します。詳しくは smbmount コマンドのマニュアルを参照してください。

以後、Windows の共有フォルダ"shared"と coLinux のディレクトリ"/mnt/smb" のデータは同じものになります。

3.1.8. 特殊な場合の Windows ネットワーク設定方法

外部接続用のネットワークアドレスが 192.168.0.0/24 の場合のネットワーク設定方法です。

「ブリッジ接続」を利用する方法です。

1) コントロールパネルから「ネットワーク接続」を開きます。
2) 外部に接続しているネットワークと「TAP-Win32 adapter」というデバイス名のネットワークの二つを選択状態にします。
3) メニューの「詳細設定」から「ブリッジ接続」を選択します。

3.1.9. coLinux のネットワーク設定方法

インストール状態では DHCP が使用されますが、DHCP サーバが動作していない環境等では固定で IP アドレスを設定する必要があります。

ネットワーク設定は ifconfig コマンドで表示することができます。

```
colinux:~# ifconfig
eth0      Link encap:Ethernet  HWaddr XX:XX:XX:XX:XX:XX
          inet addr:192.168.0.151 Bcast:192.168.0.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:189 errors:0 dropped:0 overruns:0 frame:0
          TX packets:115 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24472 (23.8 KiB)  TX bytes:9776 (9.5 KiB)
          Interrupt:2

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

colinux:~#
```

図 3-9 ifconfig コマンドの実行

eth0 デバイスの IP アドレスが表示されない場合は、固定で IP アドレスを設定する必要があります。設定すべき IP アドレスですが、「ルーター接続」の場合は「TAP-Win32 adapter」のネットワークに合わせ、「ブリッジ接続」の場合は外部ネットワークに合わせます。

ここでは、以下の表の内容に設定を変更する方法を説明します。

表 3-1 ネットワーク設定

項目	設定
IP アドレス	192.168.1.100
ネットマスク	255.255.255.0
ゲートウェイ	192.168.1.1
DNS サーバ	192.168.1.1

1) coLinux 上で/etc/network/interfaces を以下のように編集します。

```
auto lo eth0
iface lo inet loopback
iface eth0 inet static
  address 192.168.1.100
  gateway 192.168.1.1
  netmask 255.255.255.0
```

図 3-10 /etc/network/interfaces ファイルの編集例

2) coLinux 上で/etc/resolv.conf を以下のように編集します。

```
nameserver 192.168.1.1
```

図 3-11 /etc/resolve.conf ファイルの編集例

3) 以下のコマンドを実行し、編集した内容でネットワーク設定を更新します。

```
colinux:~# /etc/init.d/networking restart
Reconfiguring network interfaces: done.
colinux:~#
```

図 3-12 ネットワークの再設定コマンド

3.1.10. coLinux のアンインストール

開発終了後、coLinux をアンインストールする場合、以下の手順を行ってください。

1) Uninstall.exe の実行
coLinux をインストールしたフォルダ直下にある「Uninstall.exe」を実行します。確認ダイアログでアンインストールを指定すると、アンインストールが開始されます。アンインストールが完了すると、coLinux 本体のフォルダとファイル、デスクトップのショートカットはすべて削除されます。ただし、この時点では仮想ネットワークデバイスと Debian 環境のディスクイメージは削除されず、残ったままとなります。

2) 仮想ネットワークデバイスの削除
仮想ネットワークデバイス（TAP）を削除するには、「システムのプロパティ」から「ハードウェア」を選択し、「デバイスマネージャー」を開きます。そして、デバイスツリーから「ネットワークアダプタ」を開き、「TAP-Win32 Adapter」を削除します。

図 3-13 仮想ネットワークデバイスの削除

3) ディスクイメージの削除

Debian のディスクイメージは coLinux をインストールしたフォルダにそのまま残っています。不要な場合、フォルダごとすべて削除してください。

3.2. ソフトウェアのインストール

SUZAKU の開発するために必要なソフトウェアをインストールしていきます。

SUZAKU スターターキットでは、クロス開発と呼ばれる開発手法を採用しています。クロス開発とは、ソフトウェアの開発をそのソフトウェアが動作するシステムとは異なるシステム上で開発することです。そこで、まず始めに、クロス開発を行なえるようにするためのクロス開発環境を構築します。

クロス開発環境は複数のパッケージと呼ばれるファイルとして用意されています。以下の説明に従い、作業用 PC にインストールしてください。インストール作業は必ず root 権限で行ってください。

TIPS

ご利用の SUZAKU ボードの種類によって、必要なパッケージが異なります。ご利用になっている SUZAKU をご確認のうえ適切なパッケージをインストールしてください。

ボードの種類	章
SUZAKU-S	3.2.1. と 3.2.3.
SUZAKU-V	3.2.1. と 3.2.4.

3.2.1. 共通ソフトウェアのインストール

以下の表にあるソフトウェアは、SUZAKU-S および SUZAKU-V どちらの開発にも必要なソフトウェアです。必ずインストールしてください。

表 3-2 共通で必要なパッケージ一覧

パッケージ名	バージョン	説明
file	4.12-1 以降	Determines file type using “magic” numbers
genromfs	0.5.1-3 以降	This is the mkfs equivalent for romfs filesystem
libncurses-dev	5.4-4 以降	Developer’s libraries and docs for ncurses
perl	5.8.4-8 以降	Larry Wall's Practical Extraction and Report Language
sed	4.1.2-8 以降	The GNU sed stream editor
zlib1g-dev	1.2.2-4 以降	compression library - development

インストール方法については、お使いのディストリビューションのマニュアルを参照してください。以下に、DebianGNU Linux で採用されている apt（パッケージ管理ユーティリティ）による libncurses-dev パッケージのインストール例を示します。

```
[PC ~]# apt-get install libncurses-dev
```

図 3-14 apt-get によるパッケージのインストール例

3.2.2. ダウンローダ（Hermit）のインストール

作業用 PC に「ダウンローダ(Hermit)」をインストールします。ダウンローダは SUZAKU にソフトウェアをインストールする際に使用します。インストールするファイルは、付属 CD の suzaku/bootloader ディレクトリにあります。

1) Windows の場合
付属 CD より「Hermit-At WIN32（hermit-at-win-vX.X.XX.zip)」を任意のフォルダに展開します。vX.X.XX の X 部分は任意の数字が入り、全体で Hermit-At のバージョン番号を示しています。

2) Linux の場合
付属 CD よりパッケージファイルを用意し、インストールします。必ず root 権限で行ってください。

```
[PC ~]# dpkg -i hermit-at_x.x.x_i386.deb
```

図 3-15 Hermit のインストール

3.2.3. SUZAKU-S 専用ソフトウェアのインストール

クロス開発環境パッケージは、gzip で圧縮された tar アーカイブ形式になっています。ファイルは、付属 CD の suzaku/cross-dev/microblaze ディレクトリに用意されています。作業用 PC の /usr/local/microblaze-elf-tools/ に展開してください。

```
[PC ~]$ su –
[PC ~]# mkdir -p /usr/local/microblaze-elf-tools/
[PC ~]# cd /usr/local/microblaze-elf-tools/
[PC microblaze-elf-tools]# tar zxvf microblaze-elf-tools-20060213.tar.gz
[PC microblaze-elf-tools]# ls
bin  include  info  lib  libexec  microblaze  share
[PC microblaze-elf-tools]# exit
[PC ~]$
```

図 3-16 クロス開発用パッケージ (SUKUZA-S) のインストール

次に、クロス開発環境を使いやすくするために、パッケージの実行ファイルが入っているディレクトリを環境変数 PATH に追加します。シェルによって設定方法が異なりますので、詳しくはお使いのシェルのマニュアルを参照してください。
ここでは、bash の設定例を示します。「.bashrc」ファイルに記述しておくと、次回、ログイン時にも有効になります。

```
[PC ~]$ export PATH=$PATH:/usr/local/microblaze-elf-tools/bin
[PC ~]$
```

図 3-17 環境変数 PATH の設定例

3.2.4. SUZAKU-V 専用ソフトウェアのインストール

クロス開発環境パッケージは、複数のファイルで構成されています。ファイルは、付属 CD の suzaku/cross-dev/powerpc ディレクトリにあります。ここでは、Debian GNU Linux 用パッケージを利用しますので、さらに deb ディレクトリの下にあるパッケージファイルを全てインストールしてください。

表 3-3 SUZAKU-V 用クロス開発環境パッケージ一覧

パッケージ名	バージョン	説明
binutils-powerpc-linux	2.15-5	The GNU Binary utilities
cpp-3.3-powerpc-linux	3.3.5-13	The GNU C preprocessor
g++-3.3-powerpc-linux	3.3.5-13	The GNU C++ compiler
gcc-3.3-powerpc-linux	3.3.5-13	The GNU C compiler
libc6-powerpc-cross	2.3.2.ds1-20	GNU C Library: Shared libraries and Timezone data
libc6-dev-powerpc-cross	2.3.2.ds1-20	GNU C Library: Development Libraries and Header Files
libc6-pic-powerpc-cross	2.3.2.ds1-20	GNU C Library: PIC archive library
libc6-prof-powerpc-cross	2.3.2.ds1-20	GNU C Library: Profiling Libraries
libdb1-compat-powerpc-cross	2.1.3-7	The Berkeley database routines
libgcc1-powerpc-cross	3.3.5-13	GCC support library
libstdc++5-powerpc-cross	3.3.5-13	The GNU Standard C++ Library v3
libstdc++5-3.3-dbg-powerpc-cross	3.3.5-13	The GNU Standard C++ Library v3 (debugging files)
libstdc++5-3.3-dev-powerpc-cross	3.3.5-13	The GNU Standard C++ Library v3 (development files)
libstdc++5-3.3-pic-powerpc-cross	3.3.5-13	The GNU Standard C++ Library v3 (shared library subset kit)
linux-kernel-headers-powerpc-cross	2.5.999-test7-bk-16	Linux Kernel Headers for development

クロス開発用パッケージのインストール例を図 3-18に示します。

```
[PC ~]# dpkg -i binutils-powerpc-linux_2.15-5_i386.deb
```

図 3-18 クロス開発用パッケージ(SUZAKU-V)のインストール

インストール時に依存関係でエラーになる場合は、以下のように複数のパッケージを同時に指定してください。ワイルドカードによる指定も可能です。

```
[PC ~]# dpkg -i xxx.deb yyy.deb zzz.deb
[PC ~]# dpkg -i *.deb
```

図 3-19 複数パッケージのインストール

3.3. はじめてのコンパイルと SUZAKU へのインストール

前章で構築した開発環境のテストも兼ねて、SUZAKU のソフトウェアをコンパイルしてみましょう。

SUZAKU に必要なソフトウェアをすべてまとめたものが、付属 CD に用意されています。SUZAKU スターターキットでは、ソフトウェアの変更が可能なようにソースコードの状態で配布しています。ソフトウェアはすべてまとめられ一つの圧縮ファイルになっています。圧縮ファイルの展開から設定、コンパイルと順番に説明していきます。コンパイルの後は、インストールです。SUZAKU では PC とは異なったインストール方法が必要ですが、とても簡単に行なうことができます。

注意

作業ミスにより誤って作業用 PC 自体の OS を破壊しないために、すべての作業は一般ユーザで行なってください。root ユーザでの作業は絶対に行なわないでください。

3.3.1. ソースコードの入った圧縮ファイルを展開する

付属 CD の suzaku/dist ディレクトリに `uClinux-dist-YYYYMMDD-suzakuX.tar.gz` というファイル名のソースコードアーカイブがあります（`YYYYMMDD` はバージョンを示す年月日、`X` は SUZAKU 用に変更した履歴を表す数字です）。このファイルを任意のディレクトリに展開します。ここでは、ユーザのホームディレクトリ(~/)に展開することにします。

```
[PC ~]$ tar zxf uClinux-dist-YYYYMMDD-suzakuX.tar.gz
```

図 3-20 dist アーカイブの展開

3.3.2. メニュー画面を使ってソースコードの設定を行なう

展開されたソースコードを SUZAKU 用に設定します。ソースコードは多数の製品で使えるように構成されているため、どの製品用にコンパイルするのか設定する必要があります。

```
[PC ~/]$ cd uClinux-dist-YYYYMMDD-suzakuX
[PC ~/uClinux-dist-YYYYMMDD-suzakuX]$ make menuconfig
```

図 3-21 メニュー画面の表示

図 3-22が表示されない場合は、もう一度「3.2. ソフトウェアのインストール」を確認してください。

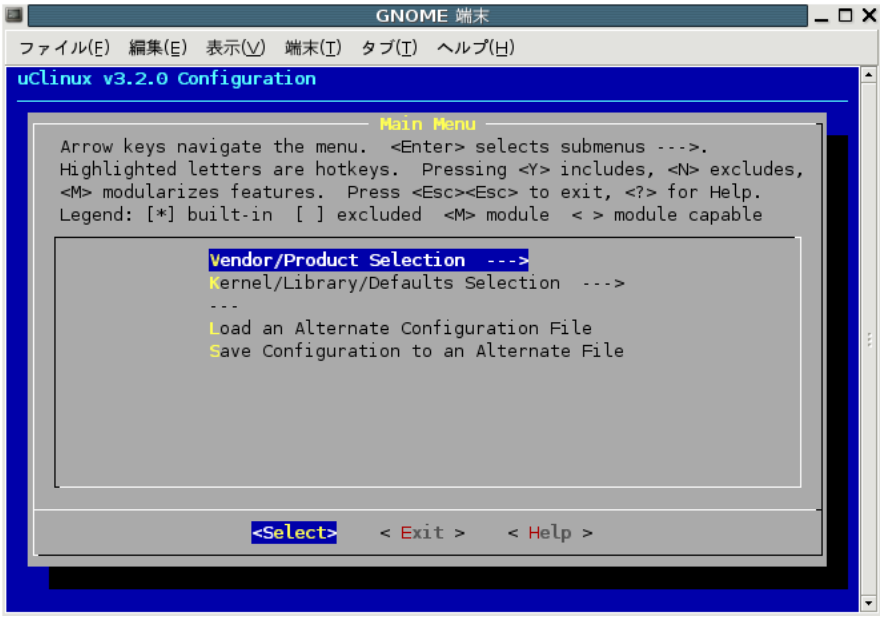

図 3-22 make menuconfig 実行直後の画面

3.3.3. メニュー画面の基本的な操作

メニュー画面の基本的な操作について説明します。

- カーソルキーでメニュー内の移動を行います。
- Enter キーを押して、サブメニューを選択できます。サブメニューは「--->」で表示されます。
- 括弧「( )」で表示されている部分は、リストから選択する部分です。Enter キーでリスト画面に移動し、上下のカーソルキーで選択対象に移動し、Enter キーで選択します。
- かぎ括弧「[ ]」は、有効無効の選択を表します。選択されると「*」がかぎ括弧内に表示されます。

すべての設定が完了後、メインメニューで Exit を選択します。設定を保存するか尋ねられるので、<Yes>を選択して保存します。画面上に設定変更のログが表示され、コマンドプロンプトに戻ります。

3.3.4. ソースコードの初期化

それではメニューを使って SUZAKU スターターキット用にソースコードを初期化してみましょう。
メニュー画面は、図 3-22にあるように、「Main Menu[3]」から始まります。画面に表示されているとおり、ここでは「Vendor/Product Selection」と「Kernel/Library/Defaults Selection」が選択できます。
まず始めに、「Vendor/Product Selection」を行ないます。「Vendor/Product Selection」をマーク状態にして「Enter」キーを押し、ベンダー名と製品名の選択画面へ移動します。ベンダー名の選択は、「(SnapGear) vendor」をマーク状態にして「Enter」キーを押し、ベンダー名の選択画面に移動します。vendor には、「AtmarkTechno」を選択してください（図 3-23 参照）。製品名には、「SUZAKU-S.STARTER-KIT」を選択します。完了したら、<Exit>を選択して「Vendor/Product Selection」を抜けます。

[3] 画面内黄色の文字

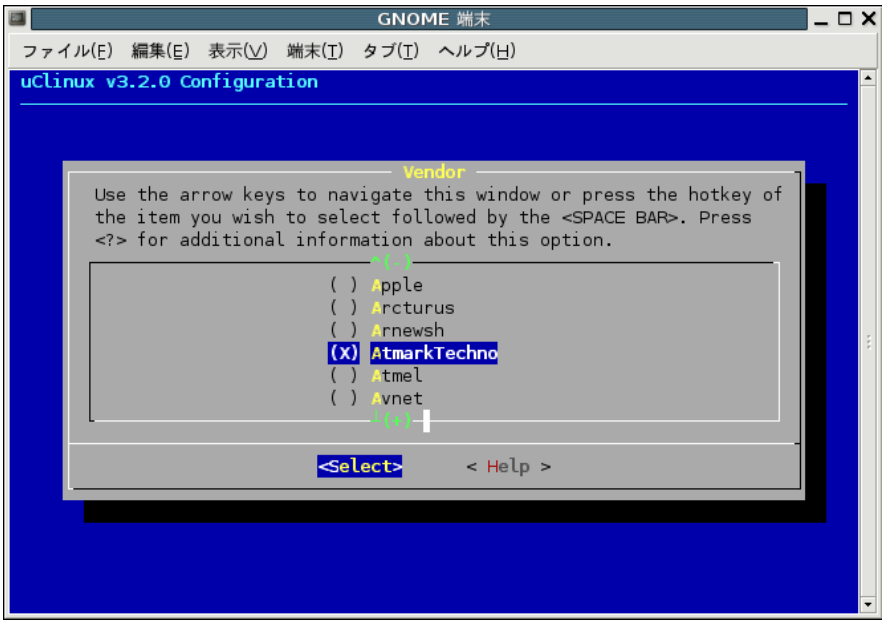

図 3-23 Vendor に AtmarkTechno を選択

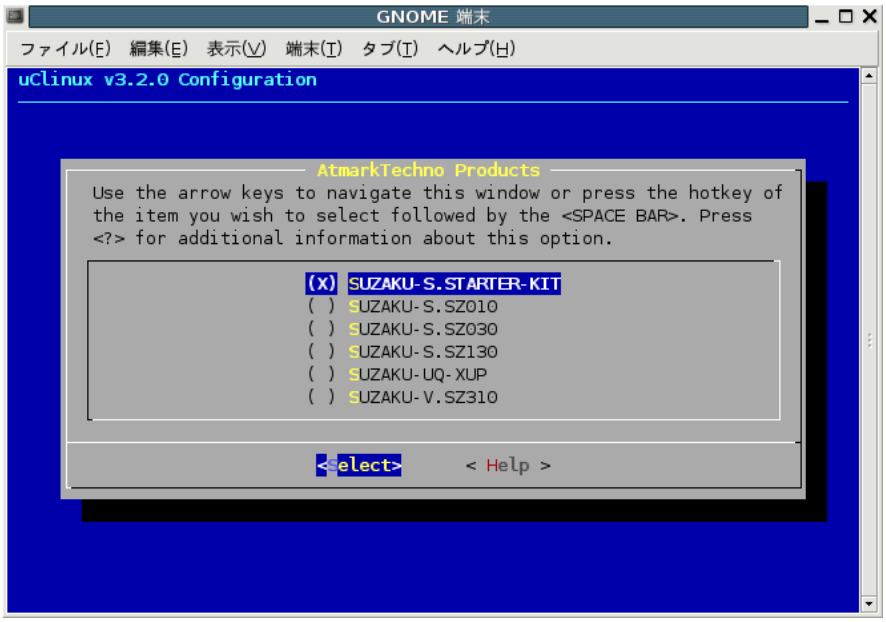

図 3-24 Product に SUZAKU-S.STARTER-KIT を選択

次に、「Kernel/Library/Defaults Selection」を選択し、「Enter」キーを押下して、カーネル/ライブラリ/デフォルト選択画面に移ります。

Kernel は自動的に選択(「---」)されています。「Libc Version」は、uClibc を選択します。異なった Libc の名前が表示されているときは、「Enter」キーを押して uClibc を選択してください。その下の 4 つの項目については、「Default all settings (lose changes)」のみを選択した状態にします。

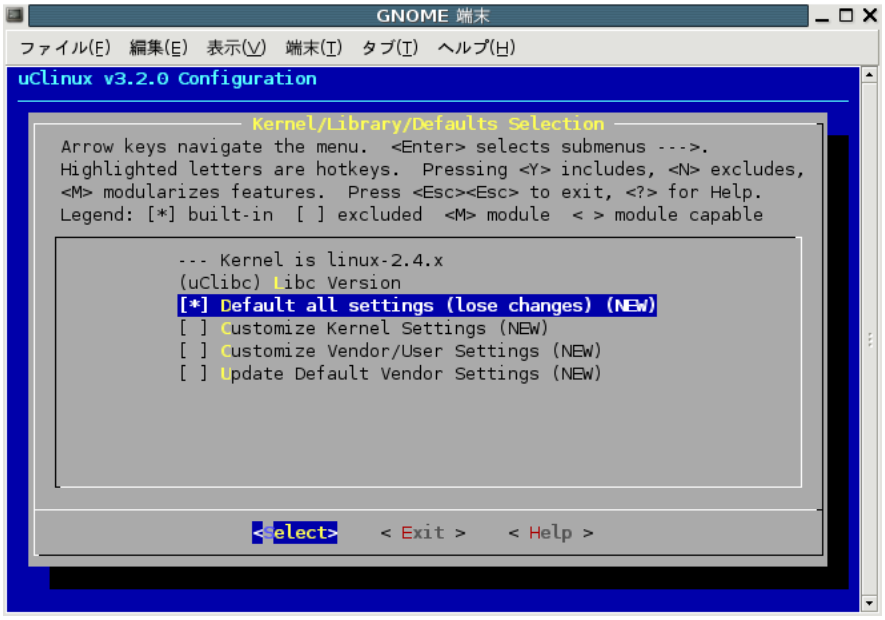

図 3-25 Default all settings を選択

完了したら＜Exit＞を選択し、「Kernel/Library/Default Selection」を抜けます。図 3-22に戻ったら、＜Exit＞を選択してコンフィギュレーションを終了します。その際に、変更したコンフィギュレーションを保存するかどうか尋ねられるので、＜Yes＞を選択します。

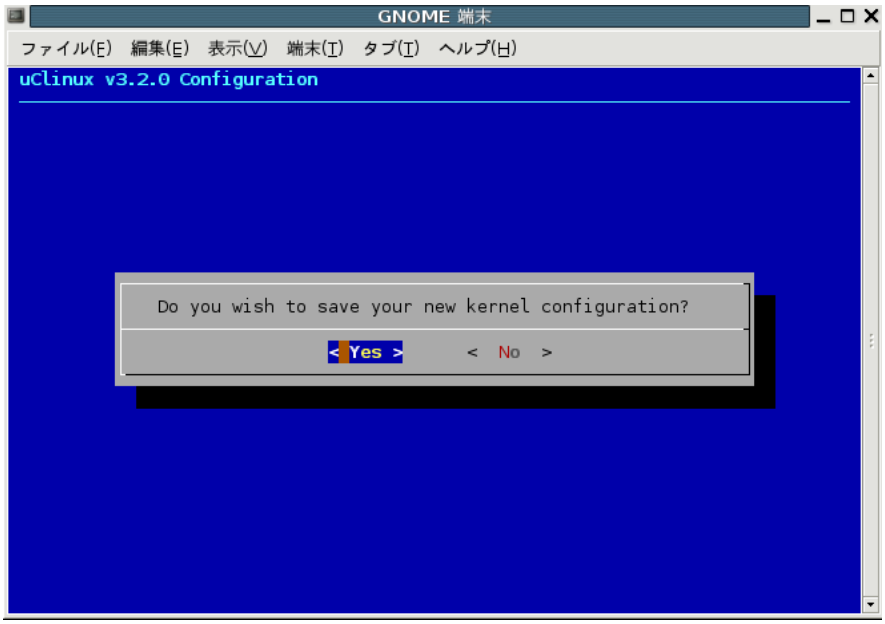

図 3-26 カーネルコンフィギュレーションを保存

質問事項が終わるとソースコードの設定が自動的に行われます。すべての設定が終わるとプロンプトに戻ります。

3.3.5. コンパイルとインストールファイルの生成

ソースコードをコンパイルし、SUZAKUスターターキットにインストールするためのファイル（フラッシュメモリのイメージファイル）を生成するには以下のコマンドを入力してください[4]。なお、この作業のことを「ビルド」と言います。

```
[PC ~/uClinux-dist-YYYYMMDD-suzakuX]$ make dep all
```

図 3-27 ビルド

TIPS

dep ターゲットは依存関係の解決を行います。2.4 系までの Linux カーネルのビルドシステムでは、make の前に依存関係の解決を行わなければなりません。2.6 系では必要ありません。

ソースコードのバージョンによっては、コンパイルの途中で一時停止し、未設定項目の問合せが表示される場合があります。通常はデフォルト設定のままで問題ありません。そのような場合はそのまま Enter キーを入力して進めてください。全ての作業が終了すると、`images` ディレクトリに、SUZAKU にインストール可能なファイル（`image.bin`）が作成されます。

```
[PC ~/uClinux-dist-YYYYMMDD-suzakuX]$ cd images
[PC ~/uClinux-dist-YYYYMMDD-suzakuX/images]$ ls
image.bin  linux.bin  romfs.img
[PC ~/uClinux-dist-YYYYMMDD-suzakuX/images]$
```

図 3-28 image.bin

3.3.6. SUZAKU へのインストール

オンボードフラッシュメモリの内容を書き換えることで、SUZAKU の機能を変更することができます。フラッシュメモリの書き換え方法を説明します。本書では、Hermit と呼ばれるツールを使った書き換え方法を紹介します。作業用 PC から SUZAKU にデータを転送することから、フラッシュメモリを書き換えることを「ダウンロード」とも言います。

3.3.7. フラッシュメモリを書き換える

それでは、「3.3.5. コンパイルとインストールファイルの生成」で作成したイメージファイルでフラッシュメモリを書き換えてみましょう。

注意

何らかの原因により「書き換えイメージの転送」に失敗した場合、SUZAKU が正常に起動しなくなる場合があります。書き換えの最中には次の点に注意してください。

- SUZAKU の電源を切らない。
- SUZAKU と開発用 PC を接続しているシリアルケーブルを外さない。

[4]全ての作成を完了するには、使用しているPCの性能によって数分から数十分程度の時間がかかります。

TIPS

SUZAKU では、フラッシュメモリの書き換え方法として、Hermit による方法の他に、

- `netflash` による書き換え
- モトローラ S レコード形式による書き換え

の２通りの方法が用意されています。詳しくは、参考文献[1]を参照ください。

まず、SUZAKUをブートローダーモードで起動します。SUZAKUのジャンパピンを以下のように設定し、SUZAKUの電源を投入します（「1.3.1. ジャンパ」参照）。

- JP1：ショート
- JP2：オープン

```
Please choose one of the following and hit enter.
a: activate second stage bootloader (default)
s: download a s-record file
t: busy loop type slot-machine
```

図 3-29 BBoot メニュー画面（スターターキット版）

ジャンパピンが正しく設定されていると、シリアル通信ソフトウェアの画面に図 3-29が表示されます。ここで「Enter」キーまたは「a」キーを押下してください。ブートローダーであるHermitの起動画面へ移ります（図 3-30）。

```
Hermit-At v1.1.3 (suzaku/microblaze) compiled at 13:49:17, Aug 15 2006
hermit>
```

図 3-30 Hermit 起動画面

次に、イメージファイルをダウンロードします。以降の手順は、作業用 PC の OS によって異なります。書き換え終了後、JP1、JP2 をオープンに設定して SUZAKU を再起動すると、新たに書き込んだイメージで起動します。

注意

シリアル通信ソフトウェアがシリアルポートを使用している状態では、Hermit がシリアルポートを使用できないためダウンロードに失敗します。必ずシリアル通信ソフトウェアを終了（シリアルポートを使用できる状態）してから Hermit を実行してください。

1) Windows の場合

「3.2.2. ダウンローダ（Hermit）のインストール」にてファイルを展開したフォルダにある、「Hermit-At WIN32（hermit.exe)」を起動します。
「Download」ボタンをクリックすると Download 画面が表示されます。

"Serial Port" には、SUZAKU と接続しているシリアルポートを設定してください。
"Image" には、書き込むイメージファイルを指定します。ファイルダイアログによる指定も可能です。
"Region" には、書き込むリージョンまたは、アドレスを指定します。ここでは「image」を選択します。

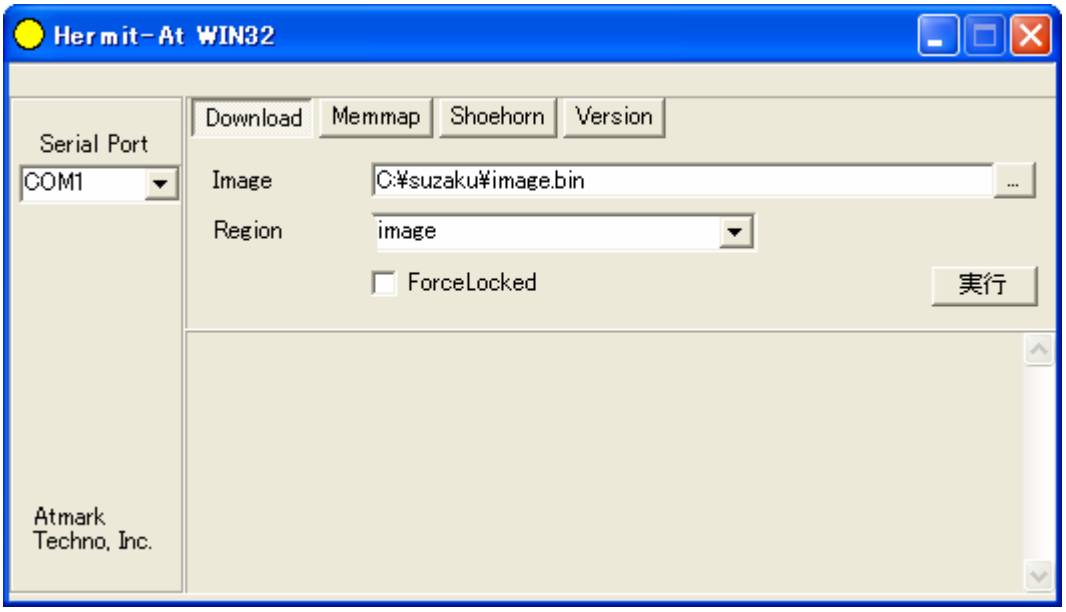

図 3-31 Download 画面

「実行」ボタンをクリックすると、フラッシュメモリへのDownloadが開始されます。Download中は、進捗状況が図 3-32のように表示されます。ダイアログは、Downloadが終了すると自動的にクローズし、図3-33のようなDownload終了画面が表示されます。

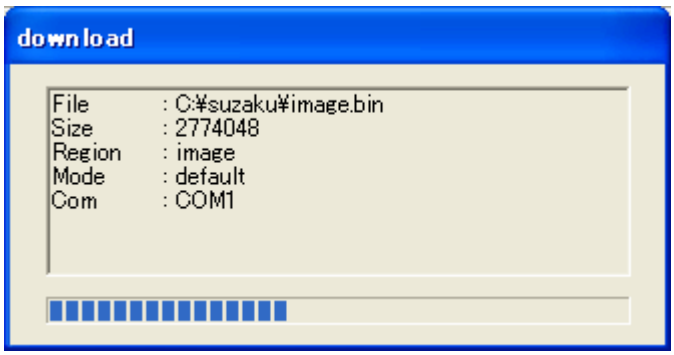

図 3-32 Download 進捗ダイアログ

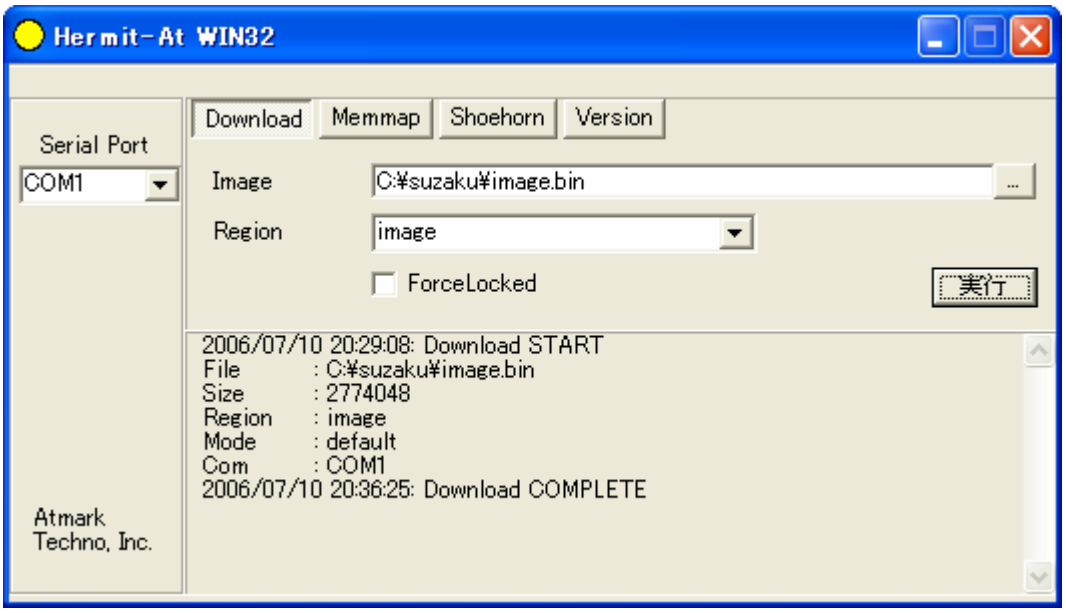

図 3-33 Download 終了画面

TIPS

FPGA およびブートローダー領域を書き換える（「Region」に「fpga」および「bootloader」を指定する）際は、「ForceLocked」をチェックする必要があります。これを選択しない場合、警告が表示されブートローダー領域への書き込みは実行されません。

2) Linux の場合

Linux が動作する作業用 PC でターミナルを起動し、イメージファイルとリージョンを指定して hermit コマンドを実行します。-i オプションでファイル名を、-r オプションでリージョン名を指定します。

```
[PC ~/uClinux-dist-YYYYMMDD-suzakuX/images/]$ hermit download –i image.bin
–r image
```

図 3-34 hermit コマンドの実行

作業用 PC で使用するシリアルポートが「ttyS0」以外の場合、オプション「--port “ポート名”」を追加してください。

TIPS

FPGA およびブートローダー領域を書き換える（-r オプションで「fpga」および「bootloader」を指定する）際は、「--force-locked」を追加する必要があります。これを指定しない場合、警告が表示されブートローダー領域への書き込みは実行されません。

4. アプリケーション開発

前章までで、SUZAKU のソフトウェア開発を行なう準備がすべて整いました。この章からは、実際に C 言語を使ってソフトウェアを作成していきます。サンプルアプリケーションを作成するために必要なソースコードは、すべて付属 CD に添付されていますので、手順通り作業を進めるだけで実際にサンプルアプリケーションを SUZAKU 上で動かすことができます。

本書ではサンプルアプリケーションの題材として、CGI を使います。理由は、

- ネットワークを使用する
- CGI についての説明が豊富

などをあげることができます。

以下では、最初に CGI の仕組みを簡単に説明します。次に実際に CGI のプログラムを C 言語で作成し、コンパイルします。最後に SUZAKU の上で実際に実行させてみます。結果は PC 上の Web ブラウザから確認することができます。

4.1. CGI

CGI とは、Common Gateway Interface の略で、動的なウェブをサービスする仕組みです。

CGI の例として、ホームページ訪問者数をカウントするものがあります。クライアント PC から WWW ブラウザでその訪問者数をカウントしているページの URL を指定すると、WWW サーバに向かってそのページのリクエストをします。リクエストされると、WWW サーバにそのホームページの HTML が読み込まれ、その中に訪問者をカウントする CGI を起動する記述が WWW サーバに CGI として解釈されます。WWW サーバは解釈した CGI の記述からプログラムを起動し、そのプログラムの処理結果を待ちます。処理が返ってきたら、その結果を読み込んだ HTML に挿入し、この例の場合訪問者数のカウントを挿入し、WWW ブラウザへレスポンスとして返します。

SUZAKU には標準で、thttpd という Web サーバが用意されています。組み込み用の小さな Web サーバですが、CGI についても対応しています。

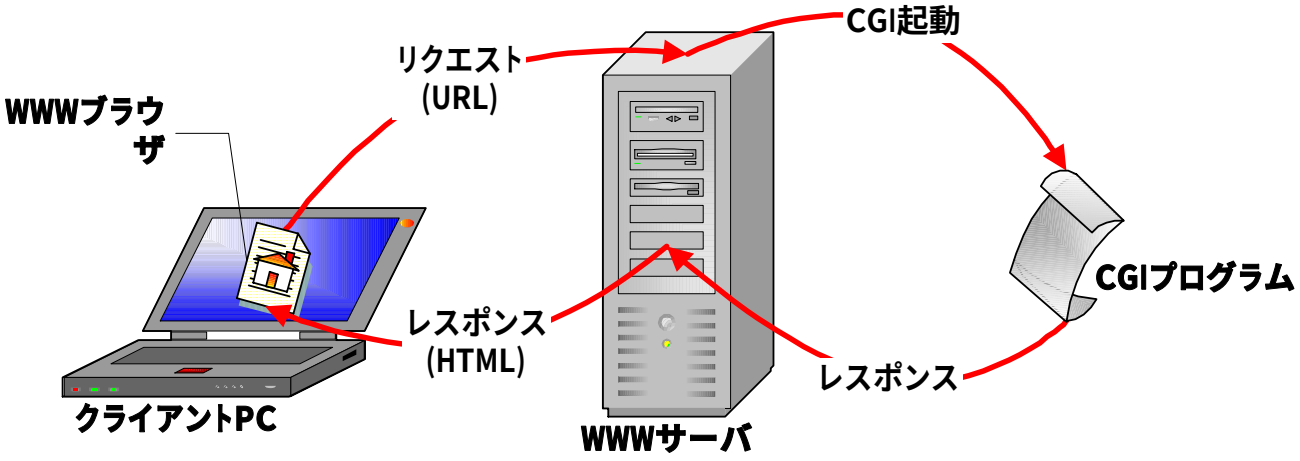

図 4-1 リクエストとレスポンス

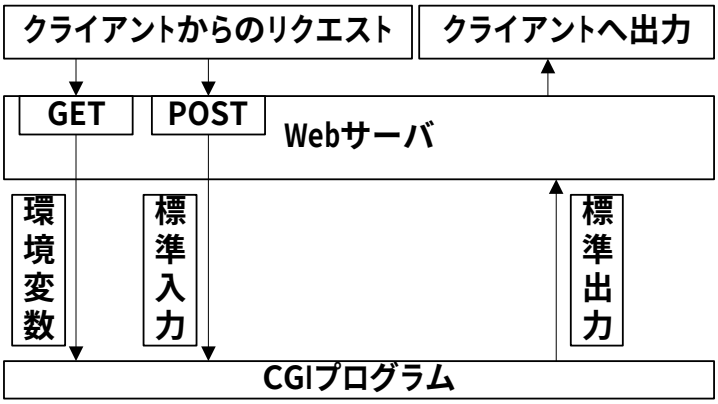

図 4-2 CGI プログラムのインターフェース

4.2. CGI プログラムの作成

それでは、実際にアプリケーションを C 言語で作成してみましょう。ここでは、Out of Tree コンパイルによる方法を説明します。Out of Tree コンパイルとは、uClinux-dist に変更を加えることなく、手軽に開発を行うことができる方法です。uClinux-dist のビルドシステムを使うため、複雑な Makefile を書く必要もありません。uClinux-dist のディレクトリ構造を木に見たて、そのディレクトリの外でコンパイルするためにこう呼ばれています。

作成したアプリケーションをuClinux-dist内に追加する方法については、参考文献[2]を参照してください。

4.2.1. サンプルプログラム

特定のテキストファイル「cgi_view.txt」の内容を出力する CGI プログラムのソースコード「cgi_view.c」を以下に示します。

```
/**
 * sample cgi application
 * Show a greet message from a specific file cgi_view.txt
 * file name: cgi_view.c
 */
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int fd;
    char buf[1000];
    int ret;

    /* コンテントタイプとヘッダー出力 */
    printf( “Content-type: text/html¥n¥n” );
    printf( “<HTML>¥n<HEAD>¥n<TITLE>cgi_view</TITLE>¥n</HEAD>¥n<BODY>¥n” );

    /* ファイル(cgi_view.txt)を読み取り専用で開く */
    fd = open( “/var/tmp/cgi_view.txt”, O_RDONLY);
    if (fd < 0) {
```

```
        printf( “open error¥n” );
        printf( “</BODY>¥n</HTML>¥n” );
        exit(1);
    }

    /* ファイルから最大 buf の大きさまで読み取り */
    ret = read(fd, buf, sizeof(buf)-1);
    buf[sizeof(buf)-1] = ‘¥0’ ;
    if (ret < 0) {
        printf( “read error¥n” );
        printf( “</BODY>¥n</HTML>¥n” );
        exit(1);
    }

    /* 読み取った文字列を本文として出力 */
    printf( “%s”, buf);
    printf( “</BODY>¥n</HTML>¥n” );

    /* ファイルを閉じる */
    close(fd);

    return  0;
}
```

図 4-3 CGI サンプルプログラム

4.2.2. Makefile

cgi_view.c用の「Makefile」を図 4-4に示します。

```
ifndef ROOTDIR
ROOTDIR=/home/atmark/uClinux-dist                                   --- ①
endif
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch

EXEC = cgi_view.cgi                                                 --- ②
OBJS = cgi_view.o                                                   --- ③

all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
        -rm -f $(EXEC) *.elf *.gdb *.o

%.o: %.c
        $(CC) -c $(CFLAGS) -o $@ $<
```

① uClinux-dist を解凍したディレクトリを指定します。
② 生成される実行ファイル名を指定します。
③ 生成される実行ファイルが依存するオブジェクトファイルを指定します。

図 4-4 CGI アプリケーション用 Makefile

4.2.3. make の実行

ホームディレクトリの下に cgi ディレクトリを作成し、上述の「cgi_view.c」と「Makefile」を用意します。さらに、CGI プログラムで表示するテキストファイル「cgi_view.txt」も用意します。次に、make コマンドを実行し、ソースコードをコンパイルします。エラーがなければ、「cgi_view.cgi」というファイルが生成されます。

```
[PC ~/]$ cd cgi
[PC ~/cgi]$ ls
Makefile  cgi_view.c  cgi_view.txt
[PC ~/cgi]$ cat cgi_view.txt
Thank you for purchasing SUZAKU. We hope you will be able to learn the
basics of using SUZAKU by completing this text.
[PC ~/cgi]$ make
  :
[PC ~/cgi]$ ls
Makefile   cgi_view.cgi      cgi_view.o
cgi_view.c  cgi_view.cgi.gdb  cgi_view.txt
```

図 4-5 CGI アプリケーションの make

4.3. CGI プログラムの実行

作成したアプリケーションを SUZAKU 上で実行するには、前の章で説明したようにイメージファイルを作成し、オンボードフラッシュを書き換えます。この方法は、SUZAKU の電源を切ったあとも変更が有効になりますが、書き換えに時間がかかりアプリケーションの開発初期やデバッグ時には不向きです。そこで、ここではアプリケーションの実行ファイルを、ftp コマンドを用いて SUZAKU ボードに転送し、実行する方法を説明します。

4.3.1. ftp によるファイル転送

作成した cgi実行ファイルを SUZAKUに転送します。ftpの際に、SUZAKUボードのIPアドレスが必要です。IPアドレスの確認は、「2.5.1. ネットワーク設定の確認」を参照してください。

```
[PC ~/cgi]$ ftp 192.168.1.100
Connected to 192.168.1.100.
220 SUZAKU-V FTP server (GNU inetutils 1.4.1) ready.
Name (192.168.1.100:atmark): root
331 Password required for root.
Password:                                    <---- パスワード(root)を入力
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /var/tmp
ftp> mput cgi_view.cgi cgi_view.txt
…
ftp> bye
[PC ~/cgi]$
```

図 4-6 ftp 転送

4.3.2. Web ブラウザによる CGI 表示

SUZAKU で起動している http サーバ thttpd は、/home/httpd/ 下が公開されています。SUZAKU では、romfs を採用しているため、起動後に /home/httpd/ にファイルを置くことができません。そのため、最初に thttpd の公開ディレクトリを変更します。それには、以下のように thttpd を再起動する必要があります。

```
[SUZAKU /]# killall thttpd
[SUZAKU /]# thttpd -c *.cgi -d /var/tmp -l /var/tmp/thttpd.log &
```

図 4-7 thttpd の再起動

転送したファイルのパーミッションを設定します。

```
[SUZAKU /]# cd /var/tmp
[SUZAKU /var/tmp]# ls
cgi_view.cgi  cgi_view.txt
[SUZAKU /var/tmp]# chmod 755 cgi_view.cgi
[SUZAKU /var/tmp]# chmod 644 cgi_view.txt
```

図 4-8 パーミッションの設定

PC のブラウザから CGI を実行します。SUZAKU の IP が 192.168.1.100 の場合、URL には `http://192.168.1.100/cgi_view.cgi` を指定します。

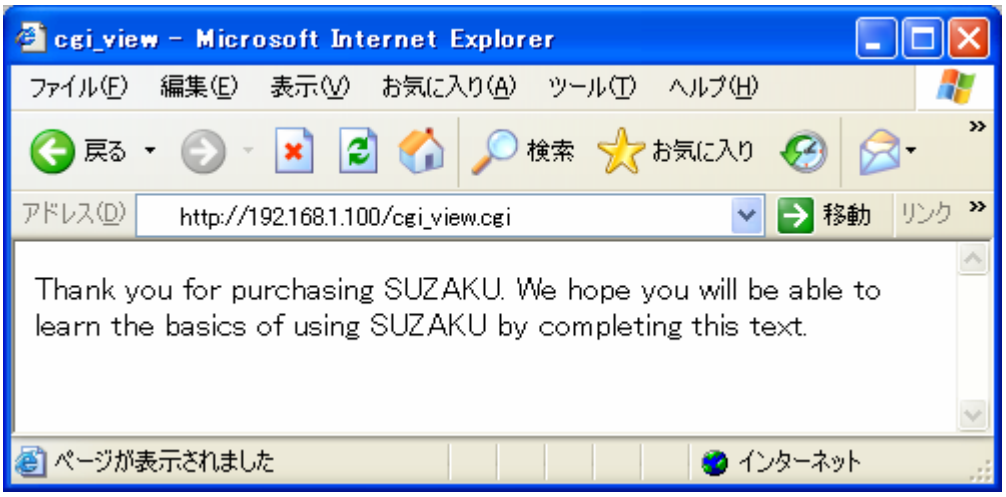

図 4-9 CGI アプリケーションの実行

5. デバイスドライバ開発

この章では、Linux のデバイスドライバを作成します。最初に Linux のデバイスドライバについて説明します。デバイスドライバの種類や使いかた、登録方法など、デバイスドライバを作成するために必要なことがらに的を絞って説明します。

デバイスドライバについて一通り解説したところで、実際にデバイスドライバを作成します。サンプルデバイスドライバのソースコードは本書に記載されていますので、手順通り行なうことにより SUZAKU 上で動作するデバイスドライバを作成することができます。また、作成したデバイスドライバをアプリケーションから使用するために、前章で作成したアプリケーションを少し変更します。

最後に作成したデバイスドライバとそのデバイスドライバを使用するように変更したアプリケーションを SUZAKU 上で実行してみます。デバイスドライバを使用する前に必要なデバイスドライバの登録作業と実際に前章で作成したアプリケーションによる確認作業を行ないます。

5.1. デバイスドライバ入門

まず始めに、Linux 上でのデバイスドライバ開発に必要な項目について説明します。

5.1.1. デバイスドライバの分類

Linux では、ディスクドライブやネットワークインターフェースといった各デバイスにアクセスする際、必ずデバイスドライバを経由します。

扱うデバイスの種類によって、デバイスドライバは三種類に分類されます。プリンタやスキャナといったようなキャラクタデータのストリームを扱うデバイスのドライバは、キャラクタデバイスドライバとして扱います。ハードディスクドライブや CD-ROM といったようなランダムアクセスが可能なデバイスのドライバは、ブロックデバイスドライバとして扱います。ネットワークインターフェースのドライバは、キャラクタ型・ブロック型どちらにも分類されません。独自にネットワークデバイスドライバとして扱います。

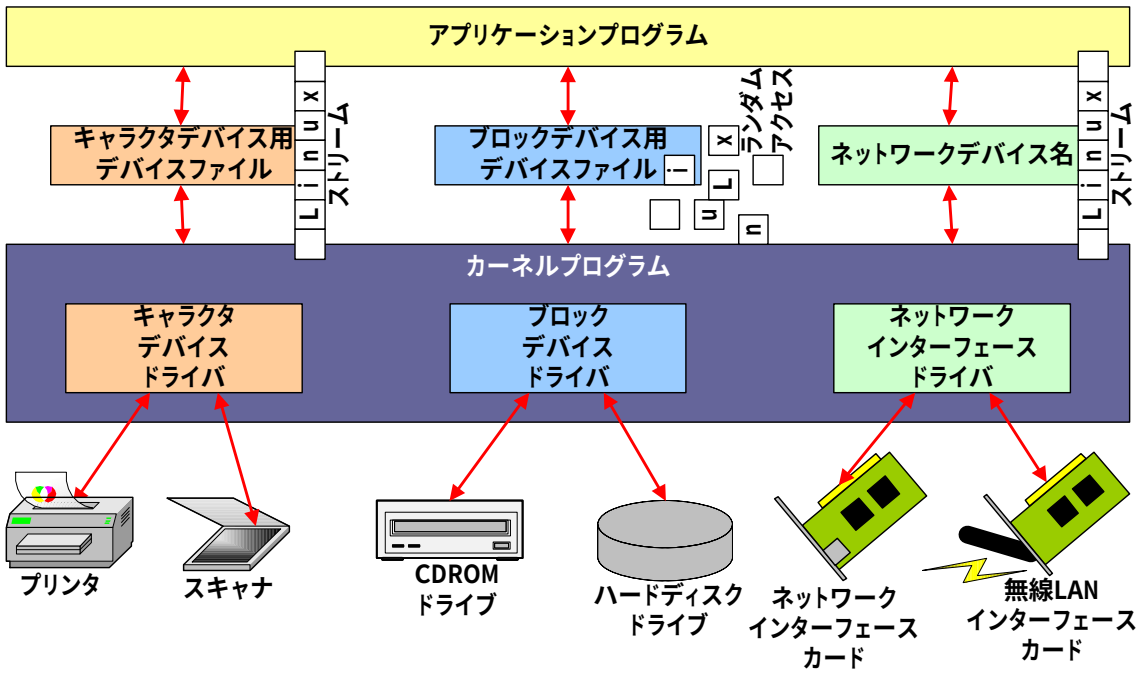

図 5-1 デバイスドライバの種類

5.1.2. デバイスファイル

Linuxでは、アプリケーションがデバイスドライバを扱うためのインターフェースとして、デバイスファイルという仕組みを利用します（/devディレクトリ下に存在するファイル群が、デバイスファイルです）。

デバイスドライバとデバイスファイルは、メジャー番号とマイナー番号によって結び付けられます。大抵は、メジャー番号がデバイスの種類を識別し、マイナー番号が同種の複数のデバイスを識別します。メジャー番号とマイナー番号はともに0から255番まであり、各デバイスドライバに割り当てられています。

一方、デバイスファイルは、ユーザーランド側（ファイルシステム上）に用意する必要があります。各デバイスファイルはすべて/devディレクトリ下に配置されます。新たなデバイスファイルを作成する場合、mknodコマンドにデバイスファイル名・メジャー番号・マイナー番号を与えて実行します。削除する場合はrmnodコマンドを用います。

5.1.3. ローダブルモジュール

Linuxのデバイスドライバは、カーネルに組み込んでしまうだけでなく、ローダブルモジュールという形でカーネルとは別のファイルとして作成することも可能です。ローダブルモジュールは、カーネル動作中に組み込んだり、取り外したりすることが可能となっています。

ローダブルモジュールを組み込むには、insmodコマンドでモジュールファイル名を指定します。以下にモジュールファイル sample.o を組み込む例を示します。

```
# insmod sample.o
```

図 5-2 insmod

ローダブルモジュールを外す場合、rmmod コマンドを使用します。insmodの際と違い、拡張子 .o はつけずにモジュール名称のみを指定します。以下にモジュール sample を外す例を示します。

```
# rmmod sample
```

図 5-3 rmmod

5.2. デバイスドライバの作成

5.2.1. サンプルドライバ

デバイスドライバと CGI のサンプルプログラムを使って、デバイスドライバで保持している文字列をブラウザで表示させてみます。

```
/**
 * Character Device Driver Sample:
 * file name: smsg.c
 */
#define __KERNEL__
#define MODULE
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/string.h>
```

```
#include <asm/uaccess.h>

static int driver_major_no = 0;
static char *msg = “Hello, everyone.”;
MODULE_PARM(msg, “s”);

/* デバイスファイルオープン時に実行 */
static int smsg_open(struct inode *inode, struct file *filp)
{
    printk(“smsg_open¥n”);
    return 0;
}

/* デバイスファイル読み取り時に実行 */
static int smsg_read(struct file *filp, char *buff, size_t count, loff_t *pos)
{
    int len;
    printk(“smsg_read: msg = %s¥n”, msg);
    len = strlen(msg);
    copy_to_user(buff, msg, len);
    return 0;
}

/* デバイスファイルクローズ時に実行 */
static int smsg_release(struct inode *inode, struct file *filp)
{
    printk(“smsg_release¥n”);
    return 0;
}

/* ファイル操作定義構造体 */
static struct file_operations driver_fops = {
    .read    = smsg_read,
    .open    = smsg_open,
    .release = smsg_release,
};

/* インストール時に実行 */
int init_module(void)
{
    int ret;
    printk(“smsg: init_module: msg = %s¥n”, msg);

    /* キャラクタ型ドライバ管理テーブルへ登録 */
    ret = register_chrdev(driver_major_no, “smsg”, &driver_fops);

    /* 登録エラー */
    if (ret < 0) {
        printk(“smsg: Major no. cannot be assigned.¥n”);
        return ret;
    }

    /* 最初に登録する場合 */
    if (driver_major_no == 0) {
        driver_major_no = ret;
```

```
        printk("smsg: Major no. is assigned to %d.¥n", ret);
    }

    return 0;
}

/* アンインストール時に実行 */
void cleanup_module(void)
{
    printk("smsg: cleanup_module¥n");

    /* キャラクタ型ドライバ管理テーブルから削除 */
    unregister_chrdev(driver_major_no, "smsg");
}
```

図 5-4 サンプルドライバ

5.2.2. サンプルドライバモジュールの Makefile

デバイスドライバモジュールの作成の場合、実行形式ファイルを作成アプリケーションとは違って、カーネルに取り込み可能なオブジェクトファイルだけを生成します。このため、リンカは使用しません。

```
ifndef ROOTDIR
ROOTDIR=/home/atmark/uClinux-dist                                    --- ①
endif
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch

OBJS = smsg.o                                                        --- ②

all: $(OBJS)

clean:
        -rm -f $(OBJS)

%.o: %.c
        $(CC) -c $(CFLAGS) -o $@ $<
```

図 5-5 サンプルドライバモジュールの Makefile

5.2.3. 改変した CGI プログラムサンプル

以前に作成した CGI プログラムをデバイスファイルから文字列を読むように改変したものが、「cgi_view2.c」です。「cgi_view.c」と同様に、make コマンドで作成することができます。

```
/**
 * sample cgi application 2
 * Show a greet message from a specific device file /var/tmp/smsg
 * file name: cgi_view2.c
 */
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int fd;
    char buf[1000];
    int ret;

    /* コンテントタイプとヘッダー出力 */
    printf("Content-type: text/html¥n¥n");
    printf("<HTML>¥n<HEAD>¥n<TITLE>cgi_view</TITLE>¥n</HEAD>¥n<BODY>¥n");

    /* ファイル(cgi_view.txt)を読み取り専用で開く */
    fd = open("/var/tmp/smsg", O_RDONLY);
    if (fd < 0) {
        printf("open error¥n");
        printf("</BODY>¥n</HTML>¥n");
        exit(1);
    }

    /* ファイルから最大 buf の大きさまで読み取り */
    ret = read(fd, buf, sizeof(buf));
    if (ret < 0) {
        printf("read error¥n");
        printf("</BODY>¥n</HTML>¥n");
        exit(1);
    }

    /* 読み取った文字列を本文として出力 */
    printf("%s", buf);
    printf("</BODY>¥n</HTML>¥n");

    /* ファイルを閉じる */
    close(fd);

    return  0;
}
```

図 5-6 改変した CGI プログラムサンプル

5.2.4. make の実行

ドライバ「smsc.c」とアプリケーション「cgi_view2.c」の 2 つをコンパイルします。ここでは、サンプルドライバモジュールの Makefile をアプリケーション用の Makefile と区別するために、Makefile.drv としています。make の際には、-f オプションで Makefile.drv を指定してください。

```
[PC ~/cgi_view2]$ ls
Makefile  Makefile.drv  cgi_view2.c  smsg.c
[PC ~/cgi_view2]$ make
[PC ~/cgi_view2]$ make -f Makefile.drv
[PC ~/cgi_view2]$ ls
Makefile      cgi_view2.c   cgi_view2.cgi.gdb  smsg.c
Makefile.drv  cgi_view2.cgi  cgi_view2.o       smsg.o
```

図 5-7 make

5.3. モジュールと CGI の実行

5.3.1. ftp によるファイル転送

SUZAKUに、「cgi_view2.cgi」と「smsg.o」をftp転送します（図 4-6参照）。

5.3.2. ローダブルモジュールの組み込みとファイル操作

デバイスドライバプログラムモジュール「smsg.o」を insmod コマンドを使ってロードします。insmod コマンド実行時に、モジュール「smsg.o」にパラメータとして msg 文字列を渡すことができます（16 文字まで）。

次に、/proc/devices を利用して、モジュールに割り当てられたメジャー番号を調べ、デバイスファイルを作成します。作成するデバイスファイルは読み取りだけ可能なキャラクタデバイスとしますので、-m のあとに 444 を指定します。

```
[SUZAKU /var/tmp]# insmod smsg.o msg=Good_Afternoon
Using smsg.o
smsg: init_module: msg = Good_Afternoon
[SUZAKU /var/tmp]# cat /proc/devices
Character devices:
  1 mem
  2 pty
  :
254 smsg

Block devices:
  :
[SUZAKU /var/tmp]# mknod -m 444 smsg c 254 0
```

図 5-8 モジュールのロードと mknod

5.3.3. Web ブラウザによる CGI 表示

PC のブラウザから CGI を実行します。SUZAKU の IP が 192.168.1.100 の場合は、URL には `http://192.168.1.100/cgi_view2.cgi` を指定します。モジュール組み込み時に `msg` パラメータで指定した文字列が表示されます。

図 5-9 CGI アプリケーションの実行 (ドライバ編)

6. SUZAKU のドライバを使ってみる

前章までは、CGI を題材にソフトウェア開発について説明してきました。これは、分かりやすさを優先させ、仮想のデバイスを使い SUZAKU 単体でも体験できるものでした。組み込み開発では、何らかのデバイスを操作することがよく行なわれます。そこで、この章では、実デバイスとして SUZAKU スターターキットに搭載されている LED を操作してみます。

6.1. SUZAKU スターターキット付属デバイスドライバについて

前章では仮想的なデバイスドライバを作成してみましたが、SUZAKU スターターキットに搭載されている LED とスイッチについては、始めからデバイスドライバが用意されています。ここではこのドライバを操作して、実際に LED を変化させたりスイッチの状態を読み取ったりしてみましょう。

6.1.1. 用意されているデバイスドライバ

これから書き込むフラッシュイメージには、以下のデバイスについてドライバが用意されています。

- 単色 LED
  ボード上に 4 個実装されている単色 LED(緑)を点けたり消したりすることができます。
- 7 セグメント LED
  ボード上に 3 個実装されている 7 セグメント LED を点けたり消したりすることができます。
- 押しボタンスイッチ
  ボード上に 3 個実装されている押しボタンスイッチの状態を取得することができます。
- ロータリコードスイッチ
  ボード上に 1 個実装されているロータリコードスイッチの状態を取得することができます。

ここからはこれらのデバイスドライバを使用し、Linux 上で実際に単色 LED と 7 セグメント LED を操作してみます。

6.1.2. 事前準備

まず、用意されたデバイスドライバを使うためには、デバイスドライバのコンパイルとフラッシュイメージの書き換えを行なう必要があります。

デバイスドライバのコンパイル方法について説明します。「3.3. はじめてのコンパイルとSUZAKUへのインストール」で行なった作業と同じように、`make menuconfig` を実行します。「`Kernel/Library/Defaults Selection --->`」を選択し、「Enter」キーを押します。図 6-1のように「`Customize Kernel Settings`」にチェックを入れ、＜`Exit`＞します。

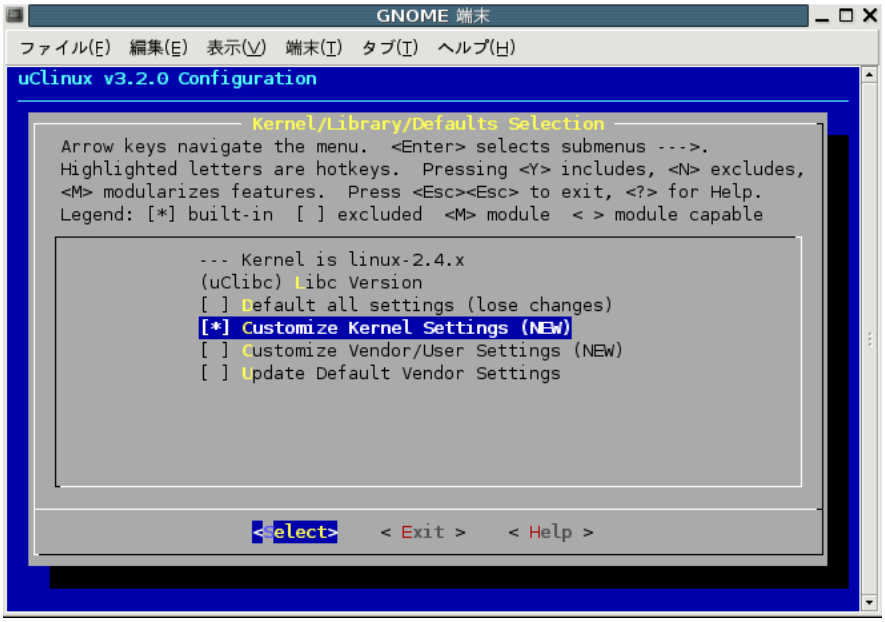

図 6-1 Customize Kernel Settings

各種設定が行なわれたあと、「Kernel Configuration」画面が表示されますので、「Character devices --->」を選択し、「Enter」キーを押します。

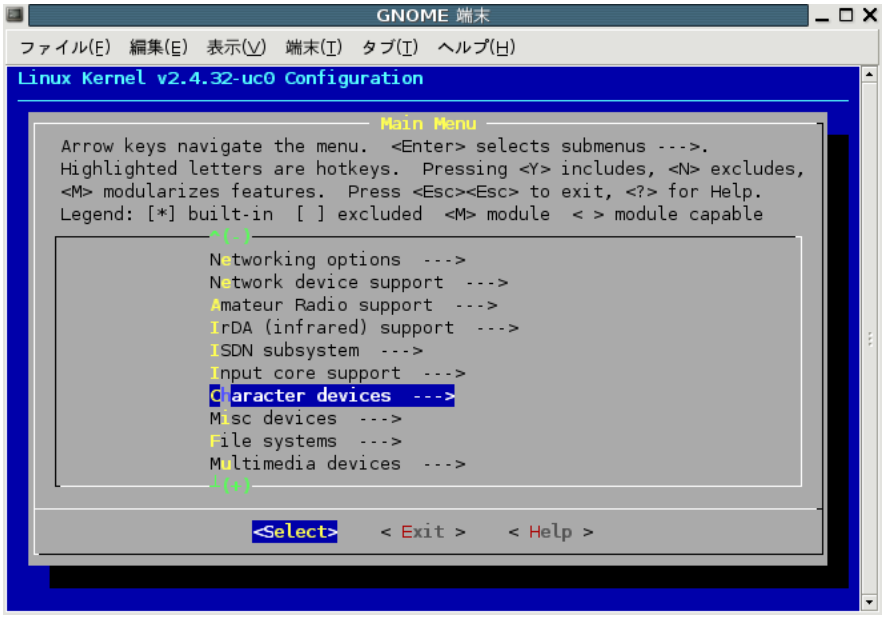

図 6-2 Character devices

キャラクタ型デバイスドライバが一覧表示された画面が表示されます。「SUZAKU I/O Board (LED/SW)」をチェック後、以下の項目にチェックを追加してください。

- LED support
- 7 segment led support
- Switch support
- Rotaly code switch support
- RS232C support

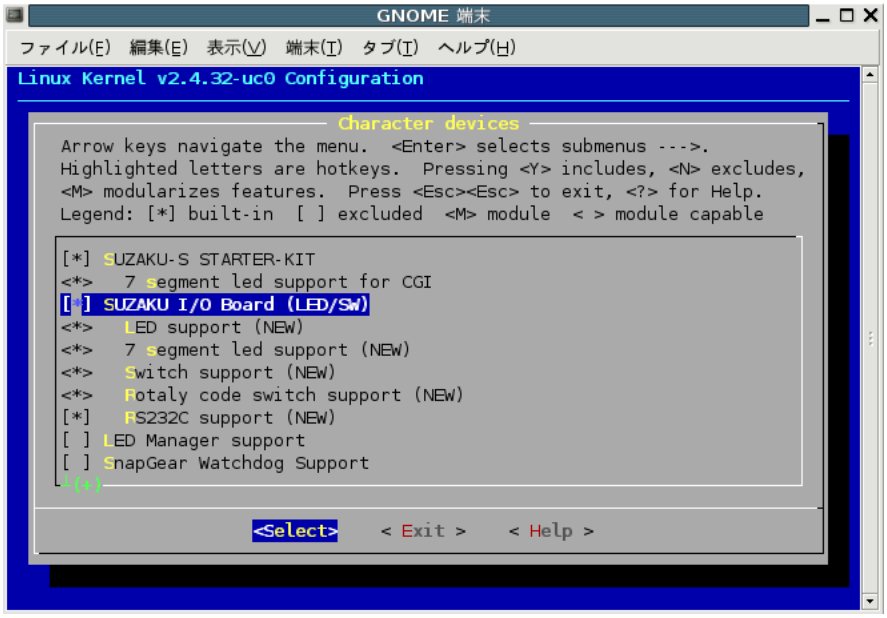

図 6-3 ドライバの追加

チェックする際に、<M>と<*>が選択できることに気づかれたでしょうか。<*>は、デバイスドライバをbuilt-inしたイメージファイルを作成するよう指示しています。<M>を指定した場合、デバイスドライバはモジュールとして作成され、イメージファイルには含まれません。ここでは、built-in形式で作成しますので、<*>形式でチェックしてください。make menuconfigが終了しましたら、ビルドを実行します（図 3-27）。

次に、フラッシュイメージの書き換えを行ないます。書き換えるフラッシュメモリのリージョンは、fpgaと image の 2 箇所になります。image リージョン用のイメージファイルは、上述のデバイスドライバを追加して再作成した image.bin です。fpga リージョン用のイメージファイルは、付属 CD の suzaku-starter-kit/image/fpga-sz130-sil-gpio_control.bin になります。SUZAKU-S スターターキット以外をお使いの方は、ファイル名の sz130 の部分がお使いの SUZAKU の型番になったものをご利用ください。

図 6-4に、Linuxの場合のフラッシュメモリの書き換え方法を示します。

```
[PC ~/uClinux-dist/images]$ hermit download -i image.bin -r image
[PC ~/]$ ls
fpga-sz130-sil-gpio_control.bin
[PC ~/]$ hermit download -i fpga-sz130-gpio_control.bin -r fpga
--force-locked
```

図 6-4 フラッシュメモリの書き換え

Windows の場合は、Hermit-At Win32 で書き換えます。fpga リージョンの書き換えは、「Region」に fpgaを選択し、「ForceLocked」にチェックを入れれば OK です。それ以外は、image リージョンの書き換えと操作は同じです。

注意

fpga リージョンの書き換えには、十分に注意してください。もし、誤ったイメージファイルで書き換えてたり、書き換え作業に失敗した場合は、ソフトウェアから修復することはできません。FPGA のコンフィギュレーションをしなおしてください。

なお、フラッシュメモリの fpga リージョンを出荷時の状態に戻したい場合は、付属 CD の suzaku-starter-kit/image/fpga-sz130.bin で再度、書き換え作業を行なってください。

6.2. Linux アプリケーションから単色 LED を操作してみる

では、実際に Linux アプリケーションからデバイスを使用してみます。まず始めに、単色 LED の場合です。

6.2.1. 単色 LED デバイスドライバ仕様

単色 LED デバイスドライバの仕様は、以下のようになっています。

表 6-1 単色 LED デバイスドライバ

ドライバ ID	sil-led
ドライバ名	SUZAKU I/O Borad -LED/SW- LED
デバイスファイル名	/dev/silled (全部)
	/dev/silled1 (D1)
	/dev/silled2 (D2)
	/dev/silled3 (D3)
	/dev/silled4 (D4)
ソースファイル所在	linux-2.4.x/drivers/char/sil-led.c

ここで大事なのは、デバイスファイル名です。Linux 上から`/dev/silled` というファイルを読み書きすることで、4 つの単色 LED を操作できるということが読み取れます。また、`/dev/silled1`～`4` を読み書きした場合、対応した LED D1～D4 を操作できるということになります。
また、write システムコールについては以下のように定義されています。

表 6-2 write システムコール (単色 LED)

書式	int write(int fd, const void *buf, size_t count);
説明	デバイスへデータを書き込みます。バッファbufから最大countバイト分のデータをデバイスへ書き込みます。 書き込みデータには、制御したい単色 LED の状態を示す文字を指定します。
引数	fd ファイルディスクリプタ buf 書き出しデータを格納するバッファ count 書き出しデータのバイト数
返り値	成功した場合は書き込んだバイト数を返し、エラーが発生した場合は-1 を返します。

`/dev/silled` に制御文字を write すれば、LED を思い通りに点けたり消したりすることができるということです。制御文字のデータフォーマットは、以下のビットマップに対応しています。

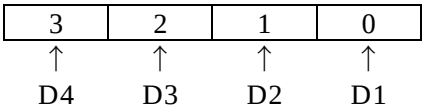

図 6-5 単色 LED のビットマップ

たとえば D1 だけを点けたい場合は”1”を、D2 だけをつけたい場合は”2”を write すればよい、ということになるわけです。

6.2.2. echo コマンドで単色 LED の状態を変更してみる

以上の仕様を踏まえて、実際に Linux 上からアプリケーションを使って単色 LED を操作してみます。Linux に用意されている echo コマンドは文字列を表示するためのものですが、リダイレクション">"とともに使うことによりデバイスファイルに対して write させる用途に使うことができるので、これを使ってみましょう。

まずは/dev/silled（全部の LED を同時に操作できる）を使って、D1 のみを点灯させてみます。echo コマンドは指定した文字に続けてリターン記号を出力してしまうのですが、-n オプションをつけることでこれを取り除くことができますので、デバイスファイルに対して write する場合、おまじないとして-n を付けるようにしてください。

```
[SUZAKU ~]# echo -n 1 > /dev/silled
```

図 6-6 silled ドライバの使用例1

ボード上の LED を見てください。D1 が点灯、D2～D4 が消灯状態になっているはずです。

次に、D2 のみを点灯させてみましょう。先ほどは 1 を書きましたが、図 6-5に従って今度は 2 をwrite します。

```
[SUZAKU ~]# echo -n 2 > /dev/silled
```

図 6-7 silled ドライバの使用例 2

D1 が消灯し、代わりに D2 が点灯します。

同じように D4 を点灯させてみます。このときは 8 を write します。

```
[SUZAKU ~]# echo -n 8 > /dev/silled
```

図 6-8 silled ドライバの使用例 3

D2 が消え D4 が点きます。

ちょっと趣向を変え、D1、D2、D4 の 3 つを点灯させてみましょう。それぞれの OR をとると 16 進数で b となりますので、これを write すればよいはずです。

```
[SUZAKU ~]# echo -n b > /dev/silled
```

図 6-9 silled ドライバの使用例 4

D1、D2、D4 が点灯、D3 が消灯状態となるはずです。

ここまでは LED 全部を扱う/dev/silled を使ってみましたが、今度はそれぞれの LED を単独で扱う /dev/silled1～4 を使用してみます。今は D3 のみが消灯していますが、これを点灯させてみます。D3 に対応するデバイスファイルは/dev/silled3 です。1 つの LED のみに対応したデバイスファイルですので、write するデータは 1 で構いません。

```
[SUZAKU ~]# echo -n 1 > /dev/silled3
```

図 6-10 silled3 ドライバの使用例 1

D3 が点灯します。このとき、D1、D2、D4 は点灯状態のまま変化せず、4 つすべての LED が点灯状態

になったと思います。このように、1 つの LED のみを扱う/dev/silled1～4 を操作した場合は、他の 3 つの LED に影響を与えません。

最後に、今点けた D3 を消してみます。/dev/silled3 に 0 を書き込みます。

```
[SUZAKU ~]# echo -n 0 > /dev/silled3
```

図 6-11 silled3 ドライバの使用例 2

D3 のみが消灯します。D1、D2、D4 には影響を与えず、点灯状態のままとなります。

6.2.3. アプリケーションを作成して単色 LED の状態を変更してみる

ここまでは echo コマンドを使いましたが、今度はプログラミング言語を使って LED を操作するアプリケーションを作ってみます。プログラム自体は今までに比べ特別難しい部分があるわけではありません。LED ドライバの仕様に従い、適切なデバイスファイルを操作するだけです。例として、D1～D4 を 1 秒ずつ順に点灯させ、最後に消灯するプログラム「silled_sample.c」と「Makefile」を作ってみます。

```
ifndef ROOTDIR
ROOTDIR=/home/atmark/uClinux-dist
endif
PATH := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch

EXEC = silled_sample
OBJS = silled_sample.o

all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
        -rm -f $(EXEC) *.elf *.gdb *.o

%.o: %.c
        $(CC) -c $(CFLAGS) -o $@ $<
```

図 6-12 単色 LED 操作サンプルプログラム用 Makefile

```
/**
 * sample application for sil-led
 * file name: silled_sample.c
 */
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char buf[2];
    int fd;
    int i, ret;

    /* デバイスファイルを読み書き可能で開く */
    fd = open("/dev/silled", O_RDWR);
    if (fd < 0) {
        printf("open error¥n");
        exit(1);
    }

    /* 文字列"1","2","4","8","0"を1秒おきに順に書く */
    for (i = 0; i < 5; i++) {
        sprintf(buf, "%d", (1 << i) & 0xf);
        ret = write(fd, buf, strlen(buf));
        if (ret < 0) {
            printf( “write error¥n” );
            exit(1);
        }
        sleep(1);
    }

    /* ファイルを閉じる */
    close(fd);

    return  0;
}
```

図 6-13 単色 LED 操作サンプルプログラム

先述の「silled_sample.c」と「Makefile」を作成したら、コンパイルしましょう。

```
[PC ~/]$ cd led_sample
[PC ~/led_sample]$ ls
Makefile  silled_sample.c
[PC ~/led_sample]$ make
  :
[PC ~/led_sample]$ ls
Makefile     silled_sample     silled_sample.c     silled_sample.gdb
silled_sample.o
```

図 6-14 単色 LED 操作サンプルプログラムの make

コンパイルに成功したら、実行ファイル「silled_sample」を ftp 転送し、実行してみましょう。単色 LED が D1～D4 まで点灯すれば成功です。

```
[SUZAKU /var/tmp]# ls
silled_sample
[SUZAKU /var/tmp]# chmod 755 silled_sample
[SUZAKU /var/tmp]# ./silled_sample
[SUZAKU /var/tmp]#
```

図 6-15 単色 LED 操作サンプルプログラムの実行

6.3. アプリケーションから 7 セグメント LED を操作してみる

今度は、7 セグメント LED の方を、前項と同様に操作してみます。

6.3.1. 7 セグメント LED デバイスドライバ仕様

7 セグメント LED の仕様は、以下のようになっています。

表 6-3 7 セグメント LED デバイスドライバ

ドライバ ID	sil-7seg
ドライバ名	SUZAKU I/O Board -LED/SW- 7SEG
デバイスファイル名	/dev/sil7seg (全部)
	/dev/sil7seg1 (LED1)
	/dev/sil7seg2 (LED2)
	/dev/sil7seg3 (LED3)
ソースファイル所在	linux-2.4.x/drivers/char/sil-7seg.c

`/dev/sil7seg` で 3 つの 7 セグメント LED すべてを、`/dev/sil7seg1`～`3` で LED1～3 それぞれを独自に操作することができます。
write システムコールについては以下のように定義されています。

表 6-4 write システムコール (7 セグメント LED)

書式	int write(int fd, const void *buf, size_t count);
説明	デバイスへデータを書き込みます。バッファ buf から最大 count バイト分のデータをデバイスへ書き込みます。 書き込みデータには、対象の 7 セグメント LED の制御状態を示す文字を指定します。
引数	fd ファイルディスクリプタ buf 書き出しデータを格納するバッファ count 書き出しデータのバイト数
返り値	成功した場合は書き込んだバイト数を返し、エラーが発生した場合は-1 を返します。

書き込みデータフォーマットは少々複雑です。各 7 セグメント LED 内の 1 つ 1 つの LED(セグメントと呼びます)には、それぞれ A～G 及び DP の名称がついています。

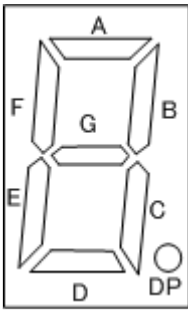

図 6-16 7 セグメント LED のセグメント

この各セグメントが、以下のビットマップに対応しています。

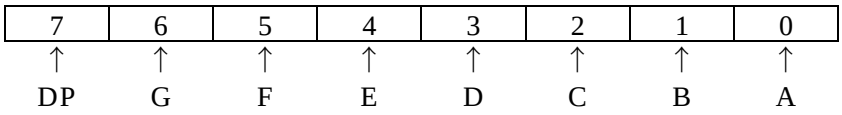

図 6-17 セグメントのビットマップ(7 セグメント LED)

ですから、0～9 を文字として 7 セグメント上に表示する場合、それぞれに対応した以下の値を write することになります。

表 6-5 数値を7セグメント LED で文字表示するための対応表

表示する文字	write する値（16 進数）
0	3F
1	6
2	5B
3	4F
4	66
5	6D
6	7D
7	27
8	7F
9	6F

`/dev/sil7seg1～3` の場合、こうして表される値がデータフォーマットとなります。`/dev/sil7seg` の場合、これにより表される値を、さらに以下のビットマップで集約したものがデータフォーマットとなります。

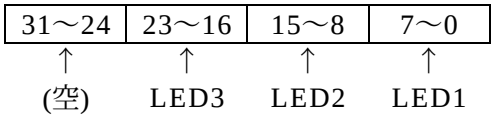

図 6-18 7 セグメント LED のビットマップ

6.3.2. echo コマンドで 7 セグメント LED の状態を変更してみる

`echo` コマンドを使って、7 セグメント LED を操作してみます。
まずは、`/dev/sil7seg` を使って、単に LED1 内のすべてのセグメントを点灯してみたいと思います。A～G 及び DP をすべて点灯させるわけですから、write する値は FF になります。

```
[SUZAKU ~]# echo -n FF > /dev/sil7seg
```

図 6-19 sil7seg ドライバの使用例 1

LED1 内のすべてのセグメントが点灯し、LED2 と LED3 はすべて消灯します。
次に、LED2 について同じことをしてみたいと思います。図 6-18に従うと、書き込むべき値はFF00 となります。

```
[SUZAKU ~]# echo -n FF00 > /dev/sil7seg
```

図 6-20 sil7seg ドライバの使用例 2

LED1 はすべて消灯すると同時に、LED2 内のすべてのセグメントが点灯します。

では今度は、”10”という文字パターンを表示してみましょう。LED2 に対し”1”を、LED1 に対し”0”を文字表示することになるので、表 6-5及び図 6-18を参照して計算すると、writeすべき値は 063Fとなります。

```
[SUZAKU ~]# echo -n 063F > /dev/sil7seg
```

図 6-21 sil7seg ドライバの使用例 3

LED1 と LED2 の表示が変わり、“10”と読める文字が表示されたと思います。
最後に、それぞれのLEDを独自に操作できる/dev/sil7seg1～3の方を使ってみたいと思います。LED3に文字パターン”2”と表示してみます。/dev/sil7seg3 に 5B を write することになります。

```
[SUZAKU ~]# echo -n 5B > /dev/sil7seg3
```

図 6-22 sil7seg3 ドライバの使用例

LED1 と LED2 の表示はそのままで LED3 のみ表示が変わり、合わせて”210”と読める表示になります。

6.3.3. アプリケーションを作成して 7 セグメント LED の状態を変更してみる

プログラミング言語を使って7セグメントLEDを操作するアプリケーションを作ってみます。例として、000 から 001、002…と 1 秒おきにカウントアップ表示していくプログラム「sil7seg_sample.c」と「Makefile」を作ってみます。

```
ifndef ROOTDIR
ROOTDIR=/home/atmark/uClinux-dist
endif
PATH := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch

EXEC = sil7seg_sample
OBJS = sil7seg_sample.o

all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
        -rm -f $(EXEC) *.elf *.gdb *.o

%.o: %.c
        $(CC) -c $(CFLAGS) -o $@ $<
```

図 6-23 7 セグメント LED 操作サンプルプログラム用 Makefile

```
/**
* sample application for sil-7seg
* file name: sil7seg_sample.c
*/
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char buf[7];
    int fd;
    int i, ret;
    const int nto7seg[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66,
                             0x6d, 0x7d, 0x27, 0x7f, 0x6f};

    /* デバイスファイルを読み書き可能で開く */
    fd = open("/dev/sil7seg", O_RDWR);
    if (fd < 0) {
        printk("open error¥n");
        exit(1);
    }

    /* 0～999 まで 1 秒置きにカウントアップしていきながら書く */
    for (i = 0; i < 1000; i++) {
        sprintf(buf, "%02x%02x%02x", nto7seg[i / 100],
                            nto7seg[(i % 100) / 10],
                            nto7seg[i % 10]);
        ret = write(fd, buf, strlen(buf));
        if (ret < 0) {
            printf( “write error¥n” );
            exit(1);
        }
        sleep(1);
    }

    /* ファイルを閉じる */
    close(fd);

    return  0;
}
```

図 6-24 7 セグメント LED 操作サンプルプログラム

上述の「sil7seg_sample.c」と「Makefile」を作成したら、コンパイルしましょう。

```
[PC ~/]$ cd 7seg_sample
[PC ~/7seg_sample]$ ls
Makefile  sil7seg_sample.c
[PC ~/7seg_sample]$ make
  :
[PC ~/7seg_sample]$ ls
Makefile    sil7seg_sample    sil7seg_sample.c    sil7seg_sample.gdb
sil7seg_sample.o
```

図 6-25 7 セグメント LED 操作サンプルプログラムの make

コンパイルに成功したら、実行ファイル「sil7seg_ sample」を ftp 転送し、実行してみましょう。7 セグメント LED が 1 秒おきにカウントアップしたでしょうか。

```
[SUZAKU /var/tmp]# ls
sil7seg_sample
[SUZAKU /var/tmp]# chmod 755 sil7seg_sample
[SUZAKU /var/tmp]# ./sil7seg_sample
```

図 6-26 7 セグメント LED 操作サンプルプログラムの実行

7. 参考文献

[1] 『SUZAKU Software Manual』,㈱アットマークテクノ.
[2] 『uClinux-dist Developers Guide』,㈱アットマークテクノ.
[3] 『make 改定版』,Andrew Oram・Steve Talbott 共著, オライリー・ジャパン.
[4] 『SUZAKU I/O Board LED/SW SIL00-U00 Software Manual』,㈱アットマークテクノ.
[5] 『LINUX デバイスドライバ（第 2 版)』,JONATHAN CORBET・ALESSANDRO RUBINI・GREG KROAH-HARTMAN 著, オライリー・ジャパン.
[6] 『Embedded UNIX vol.1』,CQ 出版社.
[7] 『Embedded UNIX vol.6』,CQ 出版社.
[8] 『TECH vol.16 組み込み Linux 入門』,CQ 出版社.
[9] 『UNIX USER 2004 年 11 月号』-意外と速い! Windows 上でそのまま起動できる coLinux-, ソフトバンク.

[10] coLinux, URL: http://www.colinux.org/
[11] GNU, URL: http://www.gnu.org/home.ja.html
[12] アットマーク・アイティ,「Linux/UNIX を知るための用語事典」,URL: http://www.atmarkit.co.jp/flinux/dictionary/indexpage/linuxindex.html

8. Appendix

8.1. ソフトウェア

SUZAKU には、Linux カーネルをベースとした OS を始め、さまざまなソフトウェアが標準で搭載されています。ここではこれらのソフトウェアの基礎的な知識や特徴について、簡単に説明します。

8.1.1. OS を使うことのメリット

Linux をベースとした OS は、PC やサーバ用途などに広く利用されています。こうした OS を使用することによるメリットはいくつもありますが、主なものについて以下に 3 つ挙げます。

- ハードウェアの抽象化

  異なるハードウェアであっても同一の機能を持つもの（例えばイーサネットデバイス）であれば、共通のインターフェースで操作することを可能にします。
  簡単にいうと、同じ Linux 用アプリケーションであれば、異なる仕様・アーキテクチャのマシン上であってもまったく同一のアプリケーションが動作するということです。デバイス自体やドライバの仕様によって一部の機能について制限が発生したり、若干の変更を加えなくてはならない場合はありますが、ほとんどの場合において共通のプラットフォームとして扱うことを可能にします。

- 資源の管理

  限られたハードウェア資源、たとえば各 I/O デバイスメモリを管理して、複数のアプリケーションからの要求による競合に対し待ち状態を発生させたり、エラーを発生させるなどの適切な処理を行うことができます。

- タスク管理による資源利用効率の向上

  複数アプリケーションの実行タスクをスケジュールして、ハードウェア資源利用の順番や時間を管理することができます。各タスクに優先順位を持たせたり、連続したハードウェア使用を制限することで、システム全体の資源利用効率を向上させます。

8.1.2. Linux の特徴

Linux は、Linus Torvalds 氏によって開発が始められたカーネルの名称です。カーネルとはその名のとおり、OS の核となる一番基本的な部分のことを指します。オープンな形での使用・開発が可能なライセンスを採用したことなどが寄与し、世界中のプログラマによって 10 年以上活発に開発が続けられています。
こうした活発な開発の成果により、Linux は他の多くの OS と比べ、格段に多種のハードウェアへの対応が行われてきています。また、多くの人々によって使用され、動作検証が行われてきたことにより、TCP/IP などといったプロトコルについての動作実績が非常に高い点も魅力といえます。

8.1.3. GNU と GPL

GNU とは、Richard Stallman 氏を創始者とするフリーソフトウェア開発プロジェクトの名称であり、世界中の開発者によるボランティア活動によって推進されています。現在は、フリーソフトウェア開発のための非営利団体であるフリーソフトウェア財団（Free Software Fouᵹncation、FSF と略す）が統括しています。
GNU の最大の特徴は、それらすべてがフリーで配布されるということです。この「フリー」とは「無料」を指す言葉ではなく、以下のように広い意味で「自由」を指します。

- ソフトウェアを複製する自由
- 使用する自由
- ソースプログラムを読む自由
- 変更する自由
- 再配布する自由

これらの自由を守るために、GNU のソフトウェアのほとんどは、GNU 一般公有使用許諾（GPL: General Public License）のライセンスに基づいて配布されています。

GPL として配布されているソフトウェアや、それらの派生物を含んだ開発物を公開する場合、ソースの公開義務や、改変使用を禁止できないなどといった GPL による制限を受けるので、注意が必要です。

なお、一般にオープンソースライセンスといった場合は、この GPL の他、BSD ライセンスや MPL（Mozilla Public License）、その他ソフトウェア固有の独自ライセンス、それらの複合などを含みます。それぞれのライセンスにより公開制限の強弱や種別などに差があるため、個別に注意深く見る必要があります。

8.1.4. GNU 開発環境

SUZAKU は、開発環境として GNU プロジェクトのツール群を採用しています。ボードとともに提供している GNU ツールについて、簡単に解説します。

binutils とは、バイナリユーティリティのことです。as（アセンブラ）や ld（リンカ）などの基本的なコマンドの他、実行ファイルやライブラリのバイナリフォーマットを扱うような様々なツールを含んでいます。

gcc は、GNU コンパイラコレクションです。GNU C コンパイラ（gcc）や GNU C++コンパイラ（g++）などが含まれます。

Glibc とは、GNU C Library のことです。C 言語の標準ライブラリや、その他 GNU 提供の多機能ライブラリを含んでいます。Glibc は現在 version2 ですが、歴史的経緯により他のバージョン体系が並立しているため libc version6（libc6）と呼ばれることもあります。

改訂履歴

Ver	年月日	改訂内容
1.0.0	2006.10.02	・初版発行
1.1.0	2006.10.20	・「はじめに」を冒頭に移動 ・「2.6. telnetログイン」を追加 ・「6.2.2. echoコマンドで単色LEDの状態を変更してみる」を追加 ・「6.3.2. echoコマンドで 7 セグメントLEDの状態を変更してみる」を追加

SUZAKU スターターキットガイド(Linux 開発編)　2006年10月20日　version 1.1.0

株式会社アットマークテクノ
060-0035 札幌市中央区北 5 条東 2 丁目 AFT ビル 6F　TEL:011-207-6550　FAX:011-207-6570