SUZAKU-S
スターターキットガイド
(Linux 開発編)

SZ130-SIL

Version 1.3.7-d308169
2009/08/03

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]

SUZAKU-S
スターターキットガイド
(Linux 開発編)

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

Version 1.3.7-d308169
2009/08/03

はじめに

SUZAKU スターターキットは、SUZAKU (朱雀) 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 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-4 FX, 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ができます。

- 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。

1. 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 (atmark-sist または 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、SUZAKU を初めて使う方や、製品付属のソフトウェアマニュアルでは難しいという開発者を対象にしています。

2. 本書の構成

本書では、SUZAKU を手にしてから開発を行うまでの手順に従い説明します。内容は 3 部構成となっています。

まず第 1 部では、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。

第 2 部では、ソフトウェア開発に必要な準備および基本操作手順を説明します。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) 。そして、Linux 開発に不可欠なディストリビューションについて解説し、コンフィグレーションおよびビルドの手順を説明します (第 4 章) 。その後、ビルドにより作成されたイメージファイルを SUZAKU のオンボードフラッシュメモリにダウンロードする方法を紹介します (第 5 章) 。

第 3 部では、実際にソフトウェア開発を体験していただきます。小規模な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 6 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 7 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 8 章) 。

3. 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。

4. 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 13. 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 ソースファイルのコード、ファイル名、ディレクトリ名など

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 14. 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

5. 謝辞

SUZAKU 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。

6. 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。

7. 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。

8. 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。下記サイトから最新版を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

図目次

表目次

1.作業の前に

この章では、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い、次に最低限必要なソフトウェアの準備を行います。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。

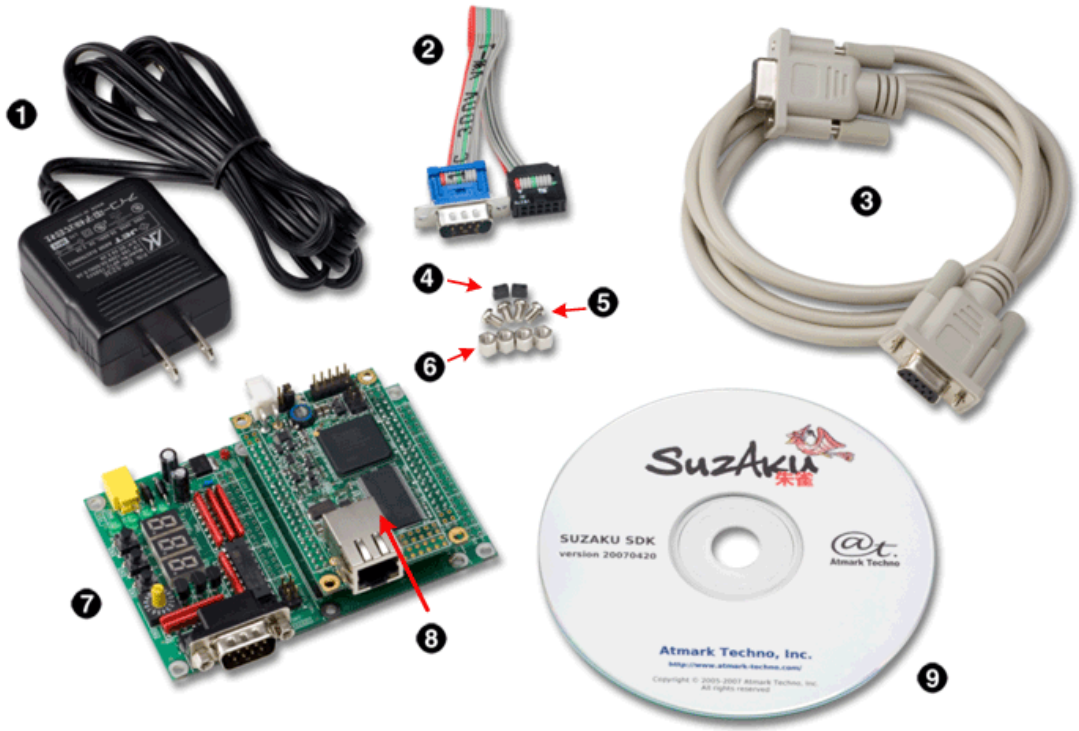

図 1.1. SUZAKU スターターキット内容物

❶ AC アダプタ 5V
❷ D-Sub9 ピン-10 ピン変換ケーブル
❸ シリアルクロスケーブル
❹ ジャンパプラグ (2 個)
❺ ネジ (4 個)
❻ スペーサー (4 個)

❼ LED/SW ボード

❽ SUZAKU (品番によって、SZ010、SZ030、SZ130 があります)

❾ CD-ROM

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、以下の条件を満たす 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	1GB 以上
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 10GB 程度以上の空き容量があること

[1] 本書では、Debian GNU/Linux を使用します。

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. SUZAKU スターターキット接続図」を参考に、シリアルクロスケーブル、電源 (AC アダプタ)、そして LAN ケーブルを SUZAKU に接続してください。

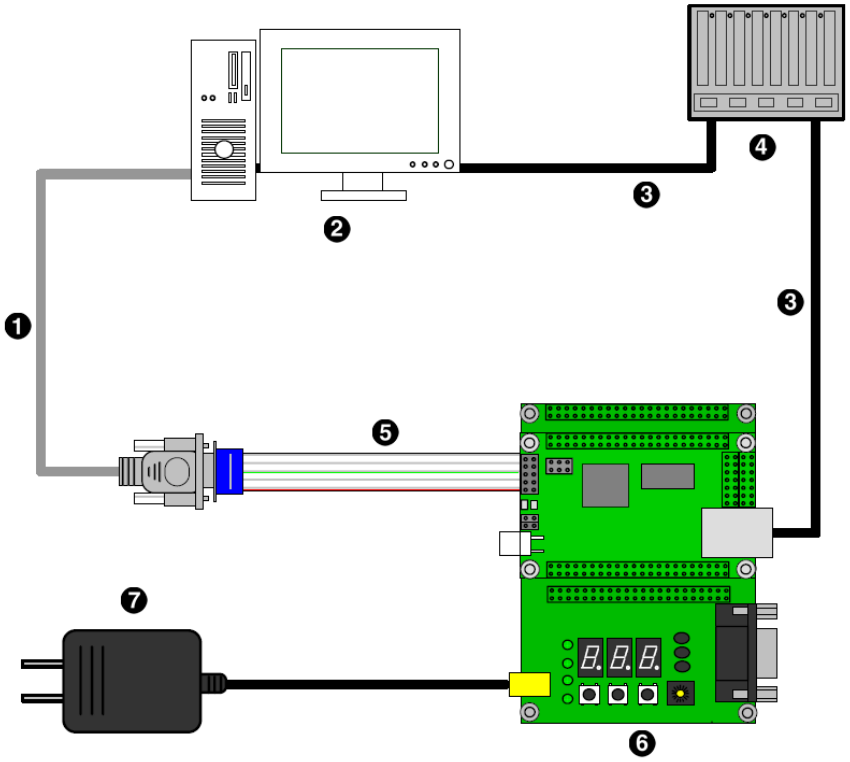

図 1.2. SUZAKU スターターキット接続図

❶ D-Sub9 ピンクロスケーブル
❷ 作業用 PC
❸ LAN ケーブル
❹ HUB
❺ D-Sub9 ピン-10 ピン変換ケーブル
❻ SUZAKU+LED/SW ボード
❼ AC アダプタ 5V

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。

   a. Tera Term Pro インストーラの実行

      setup.exe を実行します。

   b. 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。

   c. 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。

   d. 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。

   e. 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。

   f. 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。

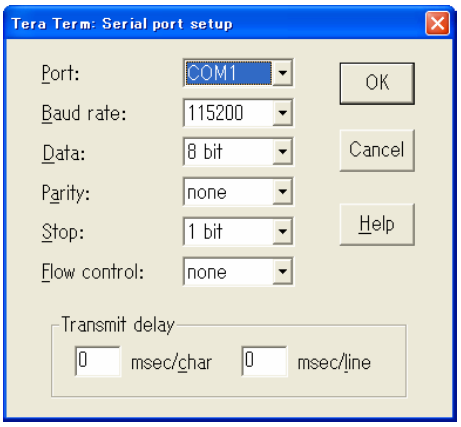

図 1.5. シリアル通信設定 (Tera Term Pro)

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. シリアル通信設定」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 キーを押下します。

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

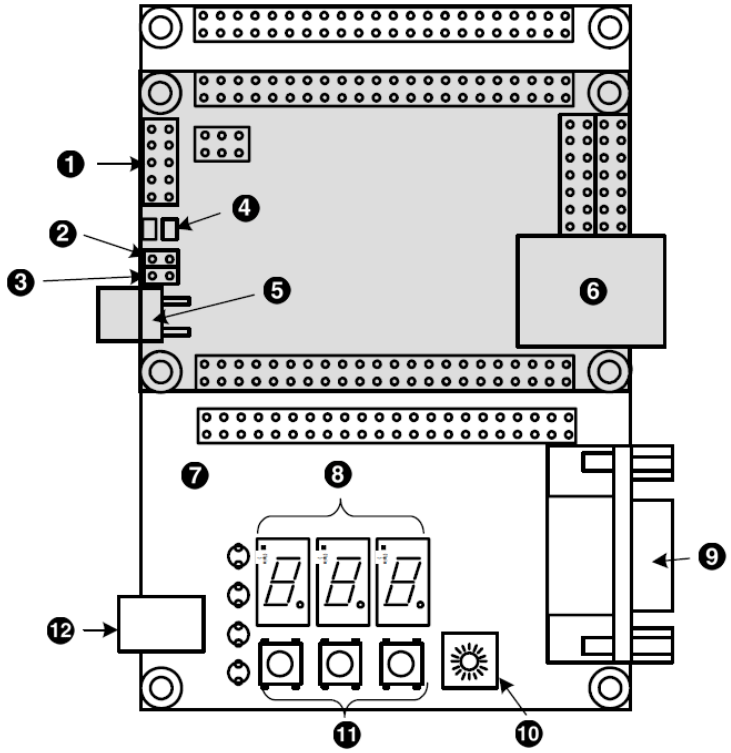

図 1.10. 各種インターフェースの配置

SUZAKU	❶	RS-232C
	❷	起動モードジャンパ (JP1)
	❸	FPGA プログラム用ジャンパ (JP2)
	❹	LED
	❺	電源入力+3.3V (LED/SW と接続時は使用しないでください)
	❻	LAN
LED/SW	❼	単色 LED
	❽	7 セグメント LED
	❾	RS-232C
	❿	ロータリコードスイッチ
	⓫	押しボタンスイッチ
	⓬	電源入力+5V

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。以下の表にジャンパの設定とその動作について示します。

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
ー	ショート	FPGA コンフィギュレーション [1]	ー

[1] 詳しくは、「SUZAKU スターターキットガイド (FPGA 開発編)」を参照してください。

ジャンパのオープン、ショートとは以下の状態を意味します。

- 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

1.4. まとめ

この章では、SUZAKU スターターキットに電源を入れる前に行う作業について説明しました。まず、ハードウェアの準備として、作業用 PC の準備および接続方法を示しました。次に、ソフトウェアの準備として、シリアル通信ソフトウェアのインストールと通信設定について説明しました。最後に、SUZAKU ボードのジャンパについて解説しました。いずれも、SUZAKU での開発には欠かせない作業です。次章へ進む前に理解し作業しておいてください。

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-suzaku6）。

```
Linux version 2.4.32-uc0 (atmark@atde) (gcc version 3.4.1 ( Xilinx EDK 8.1 Build
EDK_I.17 090206 ))
#1 Thu Dec 14 17:21:46 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... 25.39 BogoMIPS
Memory: 32MB = 32MB total
Memory: 29476KB available (988K code, 1939K 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.SZ130-SIL 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 ユーザ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.SZ130-SIL 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.SZ130-SIL 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。

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 サーバから 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。取得される IP アドレスは、DHCP サーバの設定によって異なります。

```
[SUZAKU /]# ifconfig
eth0    Link encap:Ethernet HWaddr XX:XX:XX:XX:XX:XX
        inet addr:192.168.1.100 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
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

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. IP アドレスの設定」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方法については、参考文献[1]を参照してください。

2.6. Web サーバ

thttpd [http://www.acme.com/software/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

2.7. telnet ログイン

次のユーザ名とパスワードで 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

```
[PC ~]$ telnet 192.168.1.100
Trying 192.168.1.100…
Connected to 192.168.1.100.
Escape character is '^]'.
login: root
Password:                    ←パスワードは表示されません

BusyBox v1.00 (2006.11.30-02:21+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

#
```

図 2.7. telnet ログイン

2.8. ファイル転送

FTP によるファイル転送が可能です。「表 2.3. FTP のユーザ名とパスワード」に示すユーザとパスワードでログインしてください。root ユーザの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

「図 2.8. ftp ファイル転送」に、作業用 PC から `message.txt` ファイルを SUZAKU (IP アドレス: 192.168.1.100) に FTP 転送した例を示します。

```
[PC ~]$ cat message.txt
Have fun!
[PC ~]$ ftp 192.168.1.100
200 Connected to 192.168.1.100.
220 SUZAKU-S.SZ130-SIL FTP server (GNU inetutils 1.4.1) ready.
Name (192.168.1.100:atmark): root
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /var/tmp
250 CWD command successful.
ftp> put message.txt
local: message.txt remote: message.txt
200 PORT command successful.
150 Opening BINARY mode data connection for ‘message.txt’.
226 transfer complete.
10 bytes sent in 0.00 secs (152.6 kB/s)
ftp> bye
221 Goodbyte.
[PC ~]$
```

図 2.8. ftp ファイル転送

2.9. まとめ

この章では、SUZAKU の電源を投入し実際に動かしてみました。SUZAKU の起動や終了など基本的な使用方法から、Web サーバ、telnet、ftp などのネットワーク関連のアプリケーションが利用できることを確認しました。

3.開発環境の構築

この章では、SUZAKU の開発に必要となる環境を整えます。

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 PC 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 VMware を紹介します。次に、開発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。

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、Windows 環境上に仮想的な Linux 環境を構築する方法として「VMware」を推奨します。

VMware を使用する方向けに、開発に必要なソフトウェアがインストールされた状態の OS イメージ「ATDE (Atmark Techno Development Environment)」を提供しています。初めて開発される方や、すぐに開発に着手したい場合には、ATDE の利用をお勧めします。ATDE の使用方法については、参考文献[3]をご覧ください。

3.2. クロス開発ツールのインストール

SUZAKU の開発をするために必要なソフトウェアをインストールします。

SUZAKU 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。

クロス開発ツールは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。

ATDE をご利用の方は、必要なツールはすべてインストールされています。

3.2.1. 必要な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 開発に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 3.1. 開発に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
libncurses5-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、Debian GNU/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 libncurses5-dev パッケージのインストール例を示します。

```
[PC ~]# apt-get install libncurses5-dev
```

図 3.1. 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 に搭載されたフラッシュメモリにソフトウェアを書き込む際に使用します。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_YYYYMMDD.zip`) 」を任意のフォルダに展開します。YYYYMMDD の部分は年月日を示しています。

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 3.2. Hermit のインストール

3.2.3. SUZAKU-S クロス開発パッケージのインストール

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/microblaze 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 3.2. SUZAKU-S 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-microblaze	2.10.1-mb-1	The GNU Binary utilities
cpp-microblaze	3.4.1-mb-1	The GNU C preprocessor
gcc-microblaze	3.4.1-mb-1	The GNU C compiler
elf2flt-microblaze	20070228-1	Elf2flt with PIC, ZFLAT and full reloc support.

クロス開発用パッケージのインストール例を「図 3.3. クロス開発用パッケージ (SUZAKU-S) のインストール」に示します。

```
[PC ~]# dpkg -i binutils-microblaze_2.10.1-mb-1_i386.deb
```

図 3.3. クロス開発用パッケージ (SUZAKU-S) 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 3.4. 複数パッケージのインストール

3.2.4. クロスデバッガパッケージ

クロスデバッガパッケージは、gzip で圧縮された tar アーカイブ形式になっています。作業用 PC の/usr/local/microblaze-gdb/ に展開してください。なお、このパッケージは、ATDE にも含まれておりません。

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

図 3.5. 開発用パッケージのインストール

次に、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=$PATH:/usr/local/microblaze-gdb/bin
[PC ~]$
```

図 3.6. 環境変数 PATH の設定例

3.3. まとめ

この章では、SUZAKU 用のソフトウェアを開発するための環境を整えました。まず始めに、Windows が動作している作業用 PC をお使いの方向けに、Windows 上に Linux 環境を構築する方法として、VMware を紹介しました。次に、クロス開発パッケージのインストール方法について説明しました。

4.Linux ディストリビューション

この章では、Linux を構築する上で必要不可欠な Linux ディストリビューション (以下、ディストリビューションと略記) について説明します。ディストリビューションとは、Linux カーネルとデバイスドライバ、ライブラリおよびアプリケーションをまとめたものです。現在、Debian GNU/Linux、Fedora Core など、数多くのディストリビューションが存在しています。

SUZAKU では、数あるディストリビューションの中から、uClinux-dist と呼ばれるディストリビューションをベースに独自の変更を施したものを利用します。以降では、この uClinux-dist を使って、SUZAKU へダウンロードするためのイメージファイルを作成する手順を解説します。

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ってください。root ユーザでの作業は絶対に行わないでください。

4.1. uClinux-dist について

SUZAKU では、uClinux.org が配布する uClinux 向けディストリビューションである uClinux-dist をベースに、株式会社アットマークテクノ製品に対応させた atmark-dist を採用しています。uClinux とは、MMU (Memory Management Unit) を持たないマイクロコンピュータ向けに Linux から派生した OS です。現在では、MMU を搭載したプロセッサも対応しており、uClinux 専用というわけでありません。uClinux-dist には、Linux カーネルはもちろん、ライブラリやアプリケーションがソースコードレベルで含まれています。

SUZAKU では、この uClinux-dist をベースに SUZAKU 独自の変更を加えたものを利用します。SUZAKU で使うディストリビューションは、`uClinux-dist-`*`YYYYMMDD`*`-suzaku`*`X`*`.tar.gz` というファイル名のソースコードアーカイブ形式で配布されています (付属 CD の `suzaku/dist` ディレクトリに格納されています) 。なお、ファイル名の *`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。

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

図 4.1. dist アーカイブの展開

4.2. uClinux コンフィギュレーション

ディストリビューション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を設定する必要があります。この設定作業をコンフィギュレーションと呼びます。コンフィギュレーションは、ディストリビューションを展開したディレクトリ直下で、**make menuconfig** とコマンドを入力して行います。

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

図 4.2. メニュー画面の表示

「図 4.3. make menuconfig 実行直後の画面」が表示されない場合は、もう一度「3.2. クロス開発ツールのインストール」を確認してください。

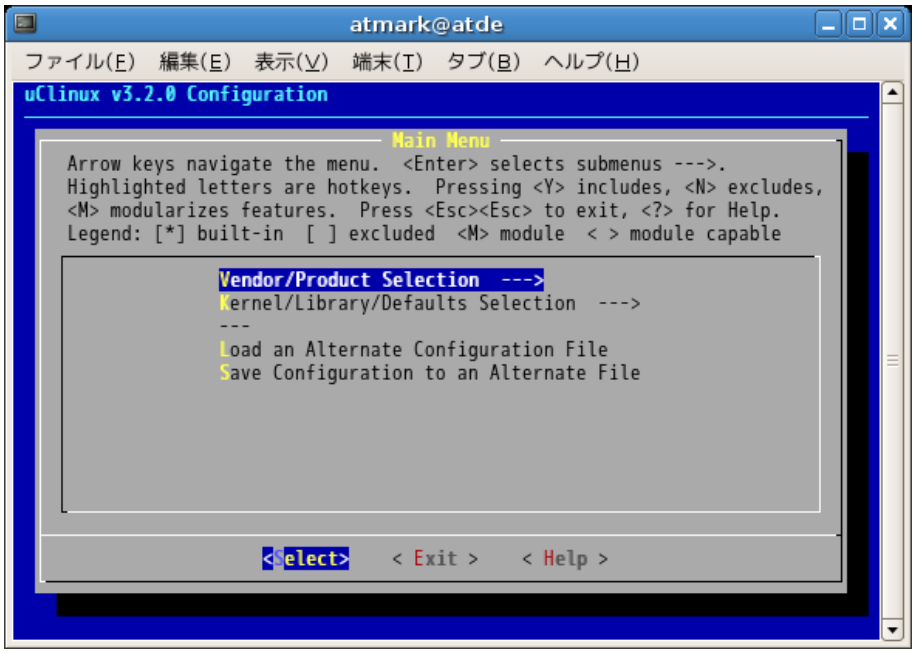

図 4.3. make menuconfig 実行直後の画面

4.2.1. 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。

4.2.2. デフォルト設定

それではメニューを使って SUZAKU スターターキットのデフォルト設定にしてみましょう。

メニュー画面は、「図 4.3. make menuconfig 実行直後の画面」にあるように、「Main Menu[1]」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、ベンダー名と製品名の選択を行います。「Vendor/Product Selection」をマーク状態にして Enter キーを押してください。「Vendor/Product Selection」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 4.4. Vendor に AtmarkTechno を選択」参照) 。製品名には、「SUZAKU-S.SZ130-SIL」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。

図 4.4. Vendor に AtmarkTechno を選択

[1] 画面内黄色の文字

図 4.5. Product に SUZAKU-S.SZ130-SIL 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。

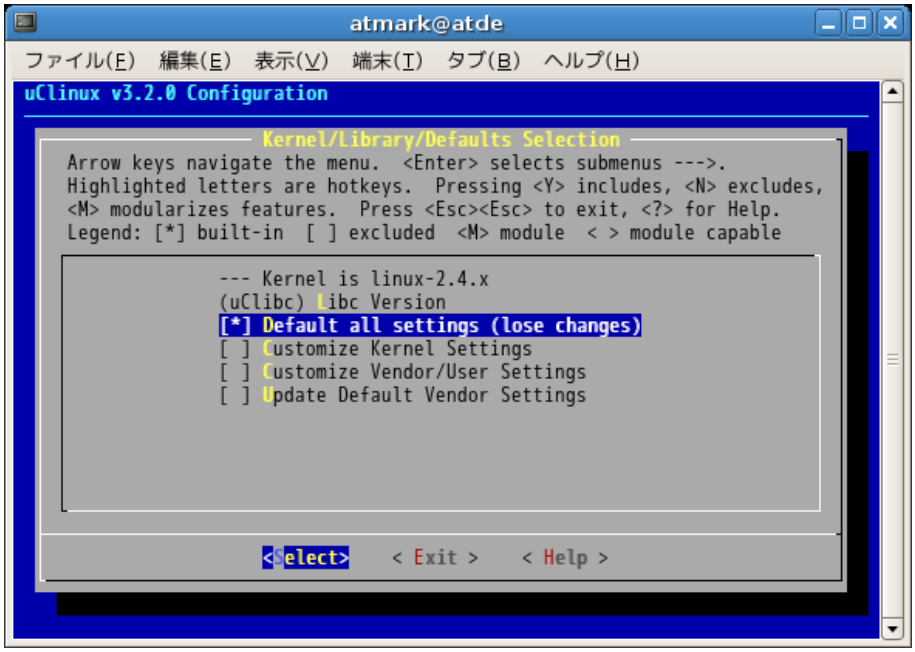

図 4.6. Default all settings を選択

完了したら< Exit >を選択し、「Kernel/Library/Default Selection」を抜けます。「図 4.3. make menuconfig 実行直後の画面」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。

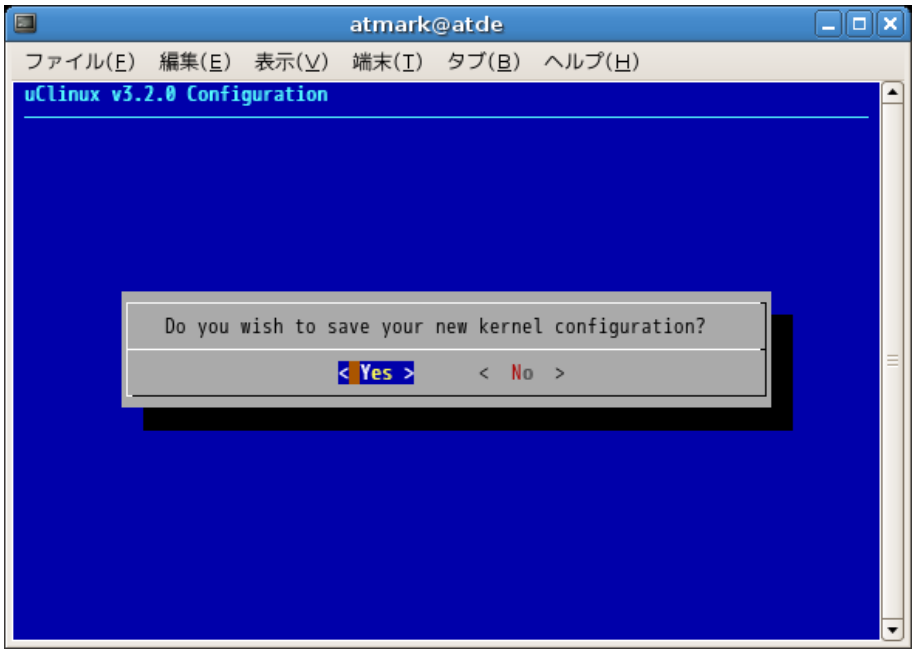

図 4.7. カーネルコンフィギュレーションを保存

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。

4.3. ビルド

コンフィギュレーションが完了したら、次は、ビルドと呼ばれる作業に移ります。ビルドとは、ディストリビューションに含まれるソースコードをコンパイルし、必要なライブラリをリンクし、SUZAKU にダウンロード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) を生成することを指します。

それでは、ビルドするために以下のコマンドを入力してください[2]。

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

図 4.8. ビルド

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。

[2] ビルドが完了するには、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。

ディストリビューション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` ディレクトリにイメージファイル (`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 4.9. image.bin

4.4. まとめ

この章では、SUZAKU で Linux を動作させるために必要な、uClinux-dist という Linux ディストリビューションについて説明しました。そして、uClinux-dist におけるコンフィギュレーションおよびビルドの手順を示し、SUZAKU スターターキット用のフラッシュメモリに書き込むイメージファイルを実際に生成しました。

ここでは、make コマンドに menuconfig というターゲットを指定したコンフィギュレーションを紹介しましたが、他にも config や xconfig といったターゲットによるコンフィグレーション方法もあります (参考文献[2]参照) 。

5.SUZAKU へのダウンロード

SUZAKU の機能を変更するには、オンボードフラッシュメモリの内容を書き換える必要があります。この章では、SUZAKU のフラッシュメモリの書き換え方法について、Hermit と呼ばれるツールを用いた方法を紹介します。なお、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言います。

5.1. フラッシュメモリを書き換える

それでは、「4.3. ビルド」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。

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。

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 5.1. 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 5.1. BBoot メニュー画面 (スターターキット版) 」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 5.2. Hermit 起動画面」) 。

```
v1.1.12 (suzaku/microblaze) compiled at 21:41:20, Oct 18 2007
hermit>
```

図 5.2. 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。

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。

5.1.1. 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。

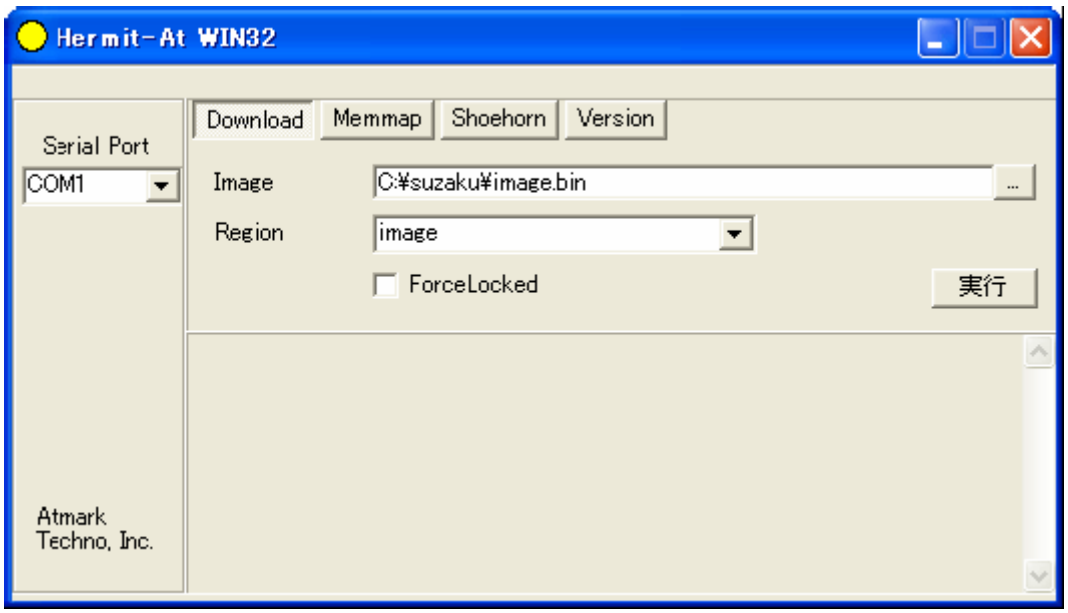

図 5.3. ダウンロード画面

「実行」ボタンをクリックすると、フラッシュメモリへのダウンロードが開始されます。ダウンロード中は、進捗状況が「図 5.4. ダウンロード進捗ダイアログ」のように表示されます。ダイアログは、ダウンロードが終了すると自動的に閉じ、「図 5.5. ダウンロード終了画面」のようなダウンロード終了画面が表示されます。

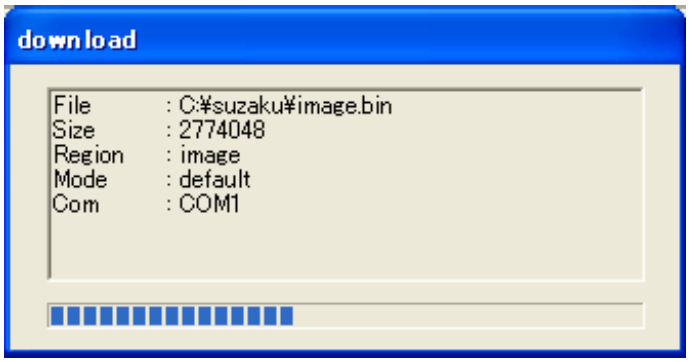

図 5.4. ダウンロード進捗ダイアログ

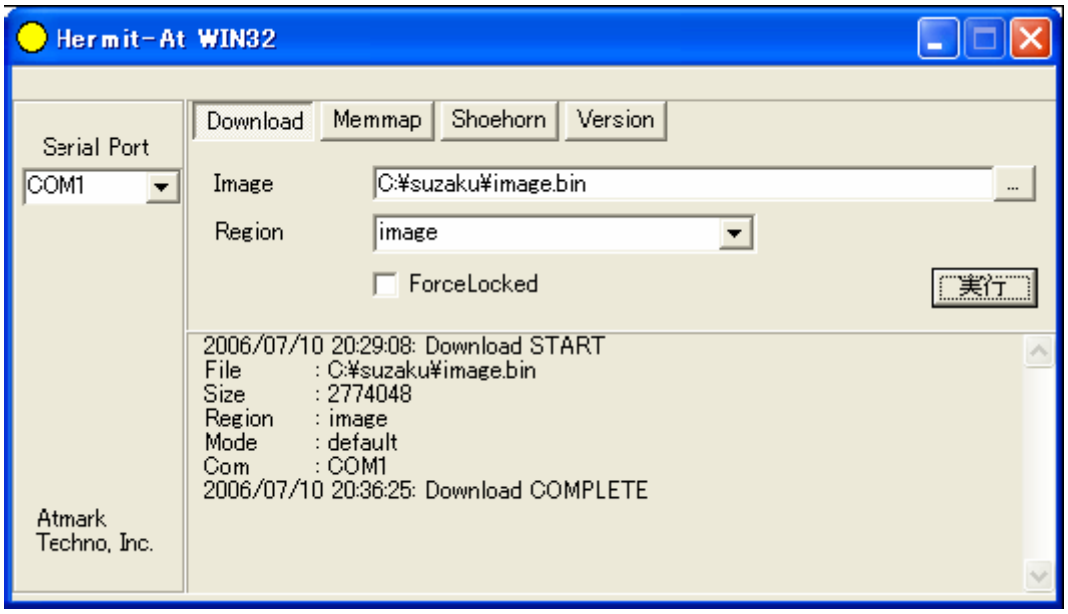

図 5.5. ダウンロード終了画面

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。

5.1.2. 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 5.6. 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。

FPGA およびブートローダー領域を書き換える (-r オプションに 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。

5.2. まとめ

この章では、オンボードフラッシュメモリの書き換え方法を紹介しました。そして、実際に Hermit と呼ばれるツールを使い、フラッシュメモリの書き換えも行いました。

SUZAKU には、ここで紹介した Hermit による方法以外にも、フラッシュメモリを書き換える方法が用意されています。詳しくは、参考文献[1]を参照してください。

6.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う基本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、すべて掲載されていますので、手順通り作業を進めるだけで実際に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、C 言語の教本に必ずと言ってよいほど掲載されている Hello World 出力プログラムと、ネットワークを使用した CGI アプリケーションを取り上げます。

6.1. Hello World

まず、最初のサンプルアプリケーションは、"Hello World" と出力する hello プログラムです。C 言語を学ばれた方ならば、1 度は試したことがあるかと思います。これを例に SUZAKU のアプリケーション開発の手順を説明します。

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` ディレクトリ構造を木 (Tree) に見たて、アプリケーションをこのディレクトリの外でコンパイルする方法です。つまり、アプリケーションのソースコードは、`uClinux-dist` ディレクトリの外に存在した状態になります。この方法では、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のが特長です。また、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。

6.1.1. コーディングとコンパイル

それでは、実際にアプリケーションプログラムを C 言語で作成してみましょう。SUZAKU での Linux 開発環境には、特定のエディタは存在しません。vi や emacs など普段使い慣れたエディタをお使いください。

ここでは、ホームディレクトリ (~/) に hello というディレクトリを作成し、その下で開発作業をすることにします。以下の `hello.c`「図 6.1. hello.c」と `Makefile`「図 6.2. Makefile (hello) 」を用意してください。`Makefile` 中の ❶ の部分は、「4. Linux ディストリビューション」で作業した `uClinux-dist` ディレクトリを指定してください。

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello World\n");
    return 0;
}
```

図 6.1. hello.c

用意ができたら、`hello.c` をコンパイルし、実行ファイル hello を作成しましょう「図 6.3. make の実行 (hello) 」。コンパイルエラーが出た場合は、エラーメッセージを参考に、`hello.c` および `Makefile` に間違いがないかを再確認してください。

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

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

EXEC = hello                             --- ❷
OBJS = hello.o                           --- ❸

all: $(EXEC)

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

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

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

❶ uClinux-dist を展開したディレクトリを指定します。

❷ 生成される実行ファイル名を指定します。

❸ 生成される実行ファイル名が依存するオブジェクトファイルを指定します。

図 6.2. Makefile (hello)

```
[PC ~/hello]$ make
:
(コンパイルメッセージ等)
:
[PC ~/hello]$ ls hello*
hello hello.c hello.gdb hello.o
```

図 6.3. make の実行 (hello)

6.1.2. 実行

作成したアプリケーションを SUZAKU 上で実行するには、イメージファイルを作成し SUZAKU 上のフラッシュメモリを書き換える必要があります。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るため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。そのため、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。

作成した実行ファイル hello を SUZAKU に FTP 転送します。FTP 転送の際には、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。ここでは、SUZAKU の IP アドレスが 192.168.1.100 の場合を「図 6.4. FTP 転送 (hello)」に示します。

```
[PC ~/hello]$ ftp 192.168.1.100
Connected to 192.168.1.100.
220 SUZAKU-S.SZ130-SIL 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> put hello
…
ftp> bye
[PC ~/hello]$
```

図 6.4. FTP 転送 (hello)

SUZAKU 側にファイル hello が転送されたことを確認します (「図 6.5. 実行パーミッションの付加と実行 (hello)」 ❶) 。次に、hello ファイルに実行パーミッションを付加してください (「図 6.5. 実行パーミッションの付加と実行 (hello)」 ❷) 。これにより、コマンドラインからファイル名を入力することでプログラムを実行できるようになります。それでは、hello を実行してみましょう (「図 6.5. 実行パーミッションの付加と実行 (hello)」 ❸) 。実行コマンドを入力する際、hello の前に「**./**」を追加してください。これは、「カレントディレクトリに存在する」という意味があります。

```
[SUZAKU /]# cd /var/tmp
[SUZAKU /var/tmp]# ls                ---❶
hello
[SUZAKU /var/tmp]# chmod 755 hello   ---❷
[SUZAKU /var/tmp]# ./hello           ---❸
Hello World
[SUZAKU /var/tmp]#
```

図 6.5. 実行パーミッションの付加と実行 (hello)

“Hello World”と出力されたでしょうか？ もし、出力されない場合には、最初から 1 ステップずつ手順を確認してください。

6.2. CGI アプリケーション

2 つ目のサンプルアプリケーションは、あるテキストファイルの内容を Web ブラウザに表示する CGI アプリケーションです。まず最初に 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 の上で実際に動作させ、PC 上の Web ブラウザから動作を確認します。

6.2.1. CGI とは

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。

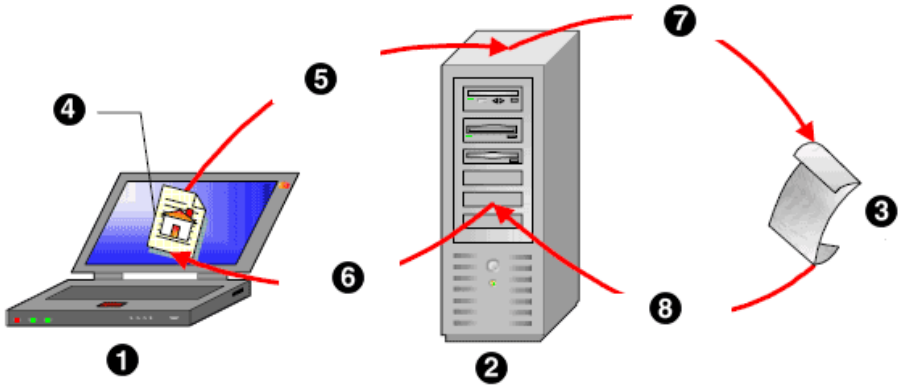

図 6.6. リクエストとレスポンス

1. クライアント PC
2. WWW サーバ
3. WWW ブラウザ
4. リクエスト(URL)
5. CGI 起動
6. CGI プログラム
7. レスポンス
8. レスポンス(HTML)

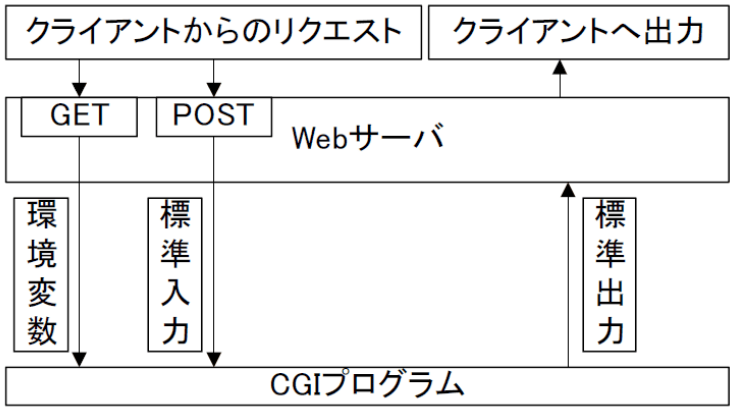

図 6.7. CGI プログラムのインターフェース

6.2.2. CGI プログラミング

それでは、CGI アプリケーションをプログラミングしてみましょう。ここでは、ホームディレクトリの下に cgi ディレクトリを作成し、`cgi_view.c`（「図 6.8. cgi_view.c」）と `Makefile`（「図 6.9. Makefile (cgi_view.cgi)」）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`（「図 6.10. cgi_view.txt」図 6-10) も用意します。

```
/**
 * 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;

    printf("Content-type: text/html\n\n");                --- ❶
    printf("<HTML>\n");
    printf("<HEAD>\n<TITLE>cgi_view</TITLE>\n</HEAD>\n<BODY>\n");

    fd = open("/var/tmp/cgi_view.txt", O_RDONLY);        --- ❷
    if (fd < 0) {
        printf("open error\n");
        printf("</BODY>\n</HTML>\n");
        exit(1);
    }

    ret = read(fd, buf, sizeof(buf)-1);                  --- ❸
    buf[sizeof(buf)-1] = '\0';
    if (ret < 0) {
        printf("read error\n");
```

```
        printf("</BODY>\n</HTML>\n");
        exit(1);
    }

    printf("%s", buf);                                          --- ❹
    printf("</BODY>\n</HTML>\n");

    close(fd);                                                  --- ❺
    return 0;
}
```

図 6.8. cgi_view.c

❶　コンテントタイプとヘッダー出力

❷　ファイル(cgi_view.txt)を読み取り専用で開く

❸　ファイルから最大 buf の大きさまで読み取り

❹　読み取った文字列を本文として出力

❺　ファイルを閉じる

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

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

EXEC = cgi_view.cgi                      --- ❷
OBJS = cgi_view.o                        --- ❸

all: $(EXEC)

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

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

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

❶　uClinux-dist を展開したディレクトリを指定します。

❷　生成される実行ファイル名を指定します。

❸　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 6.9. Makefile (cgi_view.cgi)

```
Thank you for purchasing SUZAKU. We hope you will be able to learn the basics of
using SUZAKU by completing this text.
```

図 6.10. cgi_view.txt

6.2.3. 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]$ make
:
[PC ~/cgi]$ ls
Makefile cgi_view.cgi cgi_view.o
cgi_view.c cgi_view.cgi.gdb cgi_view.txt
```

図 6.11. make の実行 (cgi_view.cgi)

6.2.4. CGI アプリケーションの実行

ここでも **hello** と同じく **ftp** コマンドを用いて SUZAKU ボードに転送し、実行することにします。まず、作成した cgi 実行ファイルおよび **cgi_view.txt** を SUZAKU に転送します (「図 6.12. FTP 転送 (cgi_view.cgi)」) 。

```
[PC ~/cgi]$ ftp 192.168.1.100
Connected to 192.168.1.100.
220 SUZAKU-S.SZ130-SIL 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 6.12. FTP 転送 (cgi_view.cgi)

転送が完了したら、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 6.13. パーミッションの設定 (cgi_view.cgi)

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 6.14. thttpd の再起動

PC の Web 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 6.15. CGI アプリケーションの実行」のように、cgi_view.txt の内容が Web ブラウザに表示されれば成功です。

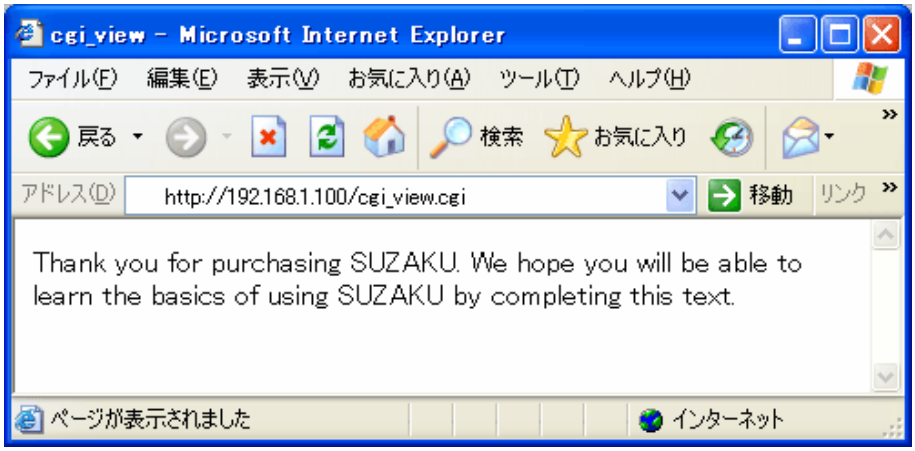

図 6.15. CGI アプリケーションの実行

6.3. まとめ

この章では、アプリケーションを開発する手順について解説しました。そして、実際に 2 つのアプリケーションを作成しました。その際、開発時に有効な Out of Tree コンパイルという手法を紹介しました。

アプリケーションの開発には、Out of Tree コンパイルの他に、uClinux-dist ディレクトリ内でコンパイルする方法もあります。実は、既に利用している chmod や thttpd、ftpd といったアプリケーションは、後者でコンパイルされています。また、これらはコンフィギュレーション時にメニューに登録され、コンパイルの実行を指定できるようになっています。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。

7.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開発を紹介します。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、モジュールの登録方法など、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解説します。

デバイスドライバについて基本事項を理解したところで、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います。

7.1. 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。

7.1.1. 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。

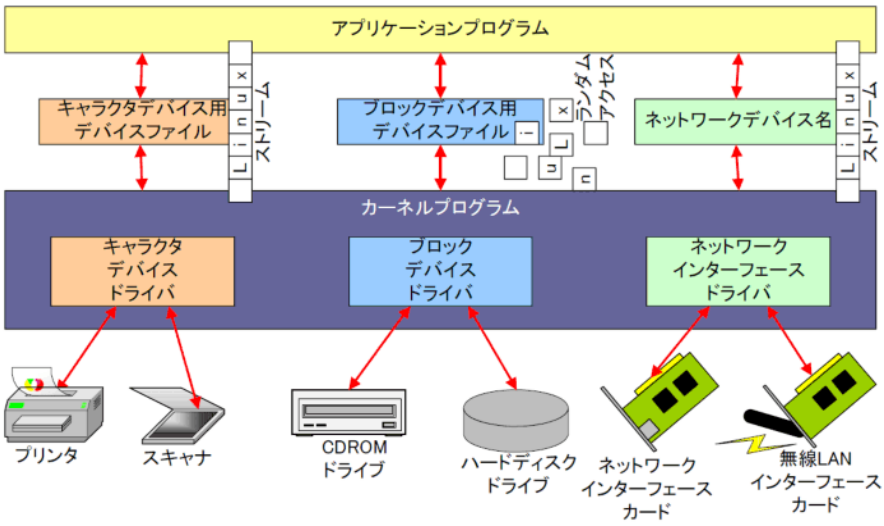

図 7.1. デバイスドライバの種類

7.1.2. 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。

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。

7.1.3. 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。

```
[SUZAKU /]# insmod sample.o
```

図 7.2. 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。

```
[SUZAKU /]# rmmod sample
```

図 7.3. rmmod

7.2. デバイスドライバの作成

7.2.1. 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
 */
#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");

static int smsg_open(struct inode *inode, struct file *filp)            --- ❶
{
    printk("smsg_open\n");
    return 0;
}

static int smsg_read(struct file *filp, char *buff, size_t count, loff_t
*pos)                                                                   --- ❷
{
    int len;
    printk("smsg_read: msg = %s\n", msg);
    len = strlen(msg);
    copy_to_user(buff, msg, len);
    return 0;
}

static int smsg_release(struct inode *inode, struct file *filp)         --- ❸
{
    printk("smsg_release\n");
    return 0;
}

static struct file_operations driver_fops = {                           --- ❹
    .read = smsg_read,
    .open = smsg_open,
    .release = smsg_release,
};

int init_module(void)                                                   --- ❺
{
    int ret;
    printk("smsg: init_module: msg = %s\n", msg);

    ret = register_chrdev(driver_major_no, "smsg", &driver_fops);       --- ❻
```

```
    if (ret < 0) {                                                    --- ❼
        printk("smsg: Major no. cannot be assigned.\n");
        return ret;
    }

    if (driver_major_no == 0) {                                       --- ❽
        driver_major_no = ret;
        printk("smsg: Major no. is assigned to %d.\n", ret);
    }
    return 0;
}

void cleanup_module(void)                                             --- ❾
{
    printk("smsg: cleanup_module\n");

    unregister_chrdev(driver_major_no, "smsg");                       --- ❿
}
```

図 7.4. smsg.c

❶ デバイスファイルオープン時に実行

❷ デバイスファイル読み取り時に実行

❸ デバイスファイルクローズ時に実行

❹ ファイル操作定義構造体

❺ インストール時に実行

❻ キャラクタ型ドライバ管理テーブルへ登録

❼ 登録エラー

❽ 最初に登録する場合

❾ アンインストール時に実行

❿ キャラクタ型ドライバ管理テーブルから削除

7.2.2. 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。

```
MODULES = smsg.o                --- ❶

ifdef UCLINUX_BUILD_KMODULE
obj-m = $(MODULES)
include $(TOPDIR)/Rules.make

else

ifndef ROOTDIR
ROOTDIR=/home/atmark/uClinux-dist                              --- ❷
endif
PATH     := $(PATH):$(ROOTDIR)/tools
UCLINUX_BUILD_KMODULE = 1
include $(ROOTDIR)/.config
include $(ROOTDIR)/config.arch

all:
        make -C $(ROOTDIR)/linux-2.4.x SUBDIRS=`pwd` modules
clean:
        -rm -f $(MODULES)

endif
```

❶ 生成されるドライバモジュールファイル名を指定します。

❷ uClinux-dist を展開したディレクトリを指定します。

図 7.5. サンプルドライバモジュールの Makefile

通常、デバイスドライバモジュールを作成する場合、__KERNEL__および MODULES を define する必要があります。しかし、ソースコード (`smsg.c`) では define していません。これは、`Makefile` (「図 7.5. サンプルドライバモジュールの Makefile」) にて define されるためで、独自に `Makefile` を用意した際は注意してください。

7.2.3. 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;

    printf("Content-type: text/html\n\n");          --- ❶
    printf("<HTML>\n");
    printf("<HEAD>\n<TITLE>cgi_view</TITLE>\n</HEAD>\n<BODY>\n");

    fd = open("/var/tmp/smsg", O_RDONLY);           --- ❷
    if (fd < 0) {
        printf("open error\n");
        printf("</BODY>\n</HTML>\n");
        exit(1);
    }

    ret = read(fd, buf, sizeof(buf));               --- ❸
    if (ret < 0) {
        printf("read error\n");
        printf("</BODY>\n</HTML>\n");
        exit(1);
    }

    printf("%s", buf);                              --- ❹
    printf("</BODY>\n</HTML>\n");

    close(fd);                                      --- ❺
    return 0;
}
```

図 7.6. 改変した CGI プログラム (cgi_view2.c)

❶ コンテントタイプとヘッダー出力

❷ ファイル(`cgi_view.txt`)を読み取り専用で開く

❸ ファイルから最大 buf の大きさまで読み取り

❹ 読み取った文字列を本文として出力

❺ ファイルを閉じる

7.2.4. make の実行

デバイスドライバ 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。

```
[PC ~/cgi_driver]$ ls
Makefile smsg.c
[PC ~/cgi_driver]$ make
[PC ~/cgi_view2]$ ls
Makefile cgi_view2.c
[PC ~/cgi_view2]$ make
```

図 7.7. make の実行

7.3. モジュールと CGI の実行

7.3.1. 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 6.12. FTP 転送 (cgi_view.cgi)」参照) 。

7.3.2. 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 7.8. モジュールのロードと mknod

7.3.3. 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 7.9. CGI アプリケーションの実行 (ドライバ編)

7.4. まとめ

この章では、Linux のデバイスドライバ開発について解説しました。まず始めに、デバイスドライバ入門としてデバイスドライバの分類やデバイスファイルについて基本的な部分を説明しました。次に仮想のデバイスドライバを作成し、モジュールの操作例を示しました。

ここでは、キャラクタデバイスドライバと呼ばれる種類のデバイスドライバのさわりを取り上げましたが、もっと詳しく、またはその他のブロックおよびネットワークデバイスドライバについて学びたい方は参考文献[6]を、作成したデバイスドライバを 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。

8.SUZAKU のドライバを使ってみる

前章までは、主に 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。

8.1. 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。

8.1.1. 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。

8.1.2. 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。「4. Linux ディストリビューション」 で行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 8.1. Customize Kernel Settings」のように「Customize Kernel Settings」にチェックを入れ、<Exit>します。

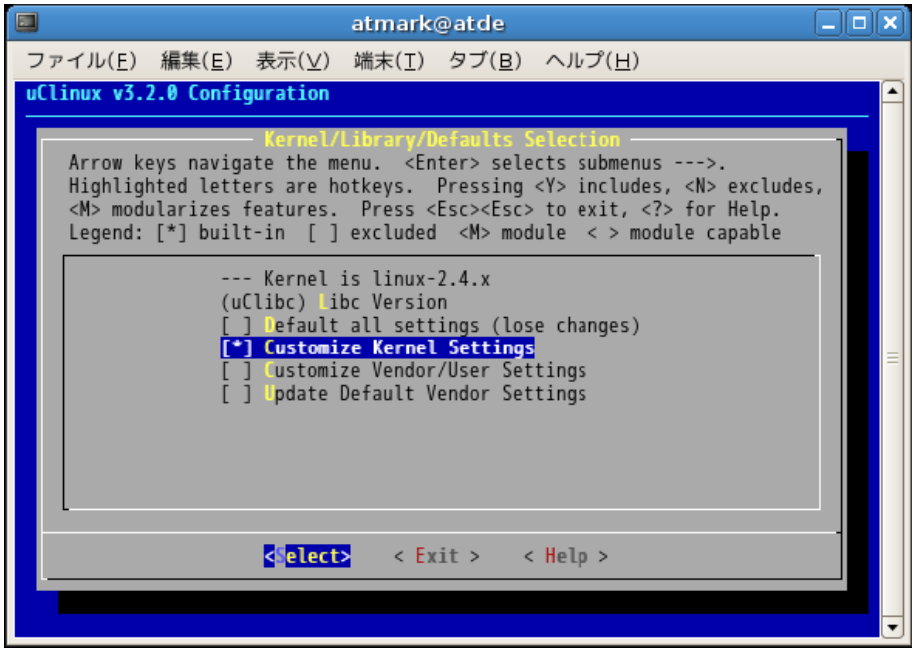

図 8.1. Customize Kernel Settings

各種設定が行われたあと、「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。

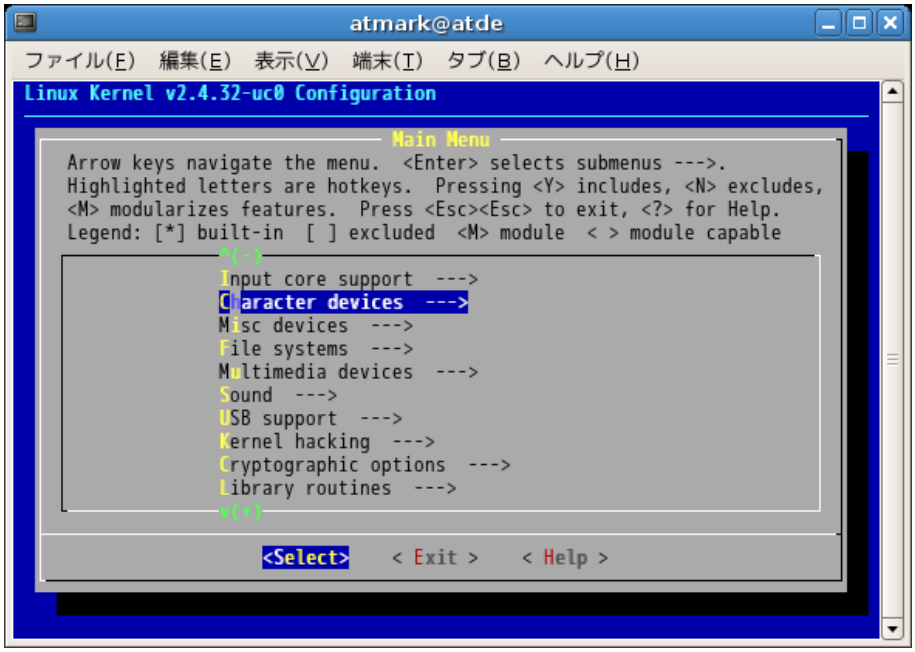

図 8.2. Character devices

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 LED/SW Board」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 8.3. 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 4.7. カーネルコンフィギュレーションを保存」。

次に、フラッシュイメージの書き換えを行います。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 8.4. フラッシュメモリの書き換え」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 8.4. 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。

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-sil.bin で再度、書き換え作業を行ってください。

8.2. 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。

8.2.1. 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 8.1. 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 8.2. 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。

3	2	1	0
↑ D4	↑ D3	↑ D2	↑ D1

図 8.5. 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。

8.2.2. 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ます。**-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 8.6. 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 8-6 に従って今度は"2"を write します。

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

図 8.7. 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 8.8. 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 8.9. 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 8.10. 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 8.11. 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。

8.2.3. 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 8.12. 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;

    fd = open("/dev/silled", O_RDWR);                    ---❶
    if (fd < 0) {
        printf("open error\n");
        exit(1);
    }
```

```
    for (i = 0; i < 5; i++) {                                   ---❷
        sprintf(buf, "%d", (1 << i) & 0xf);
        ret = write(fd, buf, strlen(buf));
        if (ret < 0) {
            printf("write error\n");
            exit(1);
        }
        sleep(1);
    }

    close(fd);                                                  ---❸
    return 0;
}
```

図 8.13. 単色 LED 操作サンプルプログラム

❶ デバイスファイルを読み書き可能で開く

❷ 文字列"1","2","4","8","0"を 1 秒おきに順に書く

❸ ファイルを閉じる

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 8.14. 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 から D4 まで順次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 8.15. 単色 LED 操作サンプルプログラムの実行

8.3. 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 を、前項と同様に操作してみます。

8.3.1. 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 8.3. 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 8.4. 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。

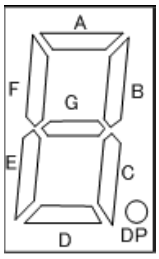

図 8.16. 7 セグメント LED のセグメント

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。

7	6	5	4	3	2	1	0
↑	↑	↑	↑	↑	↑	↑	↑
DP	G	F	E	D	C	B	A

図 8.17. 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 8.5. 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。

31 ～ 24	23 ～ 16	15 ～ 8	7 ～ 0
↑	↑	↑	↑
(空)	LED3	LED2	LED1

図 8.18. 7 セグメント LED のビットマップ

8.3.2. 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 8.19. 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 8.18. 7 セグメント LED のビットマップ」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 8.20. 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 8.5. 数値を 7 セグメント LED で文字表示するための対応表」及び「図 8.18. 7 セグメント LED のビットマップ」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 8.21. 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 8.22. 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。

8.3.3. 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 8.23. 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};

    fd = open("/dev/sil7seg", O_RDWR);          ---❶
    if (fd < 0) {
        printf("open error\n");
        exit(1);
    }

    for (i = 0; i < 1000; i++) {                ---❷
        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);
    }

    close(fd);                                  ---❸
    return 0;
}
```

図 8.24. 7 セグメント LED 操作サンプルプログラム

❶ デバイスファイルを読み書き可能で開く

❷ 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

❸ ファイルを閉じる

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 8.25. 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 8.26. 7 セグメント LED 操作サンプルプログラムの実行

8.4. まとめ

この章では、実デバイスとして LED/SW ボードに搭載されている単色 LED と 7 セグメント LED を操作するアプリケーションを開発しました。まず、各デバイスを操作するために予め用意されているデバイスドライバの仕様を確認し、コマンドラインから既存コマンドを使い操作できることを確認しました。次に、デバイスドライバを使ったアプリケーションを作成しました。

本書では、実デバイスを操作するデバイスドライバの開発は扱いませんでしたが、興味のある方は、この章で使用したデバイスドライバを読んでみるのも良いかもしれません。非常にシンプルな作りになっていますので、ガイドを一通り学習された方ならば問題なく理解できるかと思います。是非、カスタマイズにチャレンジしてください。

参考文献

[1] 「*SUZAKU ソフトウェアマニュアル*」. (株)アットマークテクノ.

[2] 「*uClinux-dist 開発者ガイド*」. (株)アットマークテクノ.

[3] 「*ATDE Install Guide*」. (株)アットマークテクノ.

[4] 「*make 改定版*」. Andrew Oram・Steve Talbott 共著, オライリー・ジャパン.

[5] 「*LED/SW Board ソフトウェアマニュアル*」. (株)アットマークテクノ.

[6] 「*LINUX デバイスドライバ (第 3 版)*」. JONATHAN CORBET・ALESSANDRO RUBINI・GREG KROAH-HARTMAN 著, オライリー・ジャパン.

[7] 「*Embedded UNIX vol.1*」. CQ 出版社.

[8] 「*Embedded UNIX vol.6*」. CQ 出版社.

[9] 「*TECH vol.16 組み込み Linux 入門*」. CQ 出版社.

[10] 「*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.

[11] *coLinux*. URL: http://www.colinux.org/.

[12] *uClinux*. URL: http://www.uclinux.org/.

[13] *GNU*. URL: http://www.gnu.org/home.ja.html.

[14] 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.

付録 A. Appendix

A.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。

A.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生成したり、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。

A.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。

A.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 Fouᴎcation、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。

A.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

バージョン	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 ・「2.7. telnet ログイン」を追加 ・「8.2.2. echo コマンドで単色 LED の状態を変更してみる」を追加 ・「8.3.2. **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
1.2.0	2006/12/15	・表示デザイン改版 ・「3.3. はじめてのコンパイルと SUZAKU へのインストール」を削除 ・「4. Linux ディストリビューション」を追加 ・「5. SUZAKU へのダウンロード」を追加 ・SUZAKU-V スターターキットに関する記述を追加
1.2.1	2007/02/16	・「7. デバイスドライバ開発」の Makefile を変更
1.2.2	2007/04/20	・「3.2.1. 必要なソフトウェアのインストール」にパッケージを追記
1.3.0	2007/10/19	・coLinux から VMware に移行 ・「3.1. Windows 上に Linux 環境を構築する」を ATDE 向けの記述に書き換え ・「3.2.3. SUZAKU-S クロス開発パッケージのインストール」を deb/tgz パッケージを使用するように書き換え ・SUZAKU-V 用の記述を削除
1.3.1	2007/12/14	・「図 4.4. Vendor に AtmarkTechno を選択」の参照エラーを修正
1.3.2	2008/02/15	・誤記修正
1.3.3	2008/07/25	・誤記修正
1.3.4	2008/09/26	・タイトルを英語表記からカタカナ表記に
1.3.5	2009/03/19	・参照先を記述する際の表記を統一 ・「1.1.1. SUZAKU スターターキットの内容物確認」の誤記を修正 ・「図 7.4. smsg.c」の誤記を修正 ・「図 8.24. 7 セグメント LED 操作サンプルプログラム」の誤記を修正 ・「7.3.1. ftp によるファイル転送」ローダブルモジュールのファイル名を修正 ・「図 8.14. 単色 LED 操作サンプルプログラムの make」ls コマンドの出力結果を修正
1.3.6	2009/07/17	・本文のレイアウト統一
1.3.7	2009/07/29	・製品保証に関する記載を http://www.atmark-techno.com/support/warranty-policy に移動(2009/08/03 適用)

SUZAKU-S スターターキットガイド (Linux 開発編)
Version 1.3.7-d308169
2009/08/03

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