

演習環境の構築

もくじ

1 章 作業の流れ

1. クロスツールチェーン	・・・	1
2. SUZAKU-V の設定	・・・	1
3. 演習前の準備	・・・	1

2 章 クロスツールチェーンの構築

1. ツールチェーンの構築に必要なもの	・・・	2
2. 作業準備	・・・	5
3. CentOS	・・・	6
4. binutils のインストール	・・・	6
5. GCC のコンパイル(1 回目)	・・・	7
6. Linux カーネルヘッダの作成	・・・	7
7. uClibc のクロスコンパイル	・・・	8
8. GCC のコンパイル(2 回目)	・・・	9

3 章 ブートローダ Hermit の書き換え

1. SUZAKU-V(出荷状態)のブートシーケンス	・・・	11
2. ブートローダ Hermit の書き換え	・・・	14
3. 準備作業	・・・	16
4. ブートローダ Hermit のコンパイル	・・・	17
5. ブートローダ Hermit の書き込み	・・・	18

4 章 FPGA 初期化データの書き換え

1. 作業準備	・・・	21
2. FPGA 初期化データの書き込み	・・・	22

5 章 授業資料の準備

1. 演習第 13 回	・・・	25
2. 演習第 14 回	・・・	27

1 章 作業の流れ

はじめに、この資料は演習環境を構築するためのものです。演習環境を構築するためには、以下のファイルが必要になります。

演習環境の構築 (本資料)

virtex405-suzakuv.dts

suzaku_led.c

これらのファイルがあることを確認して、作業を進めてください。

次に、この資料における作業の流れを簡単に説明します。詳しい説明は各章に記述してありますので参考にしてください。

1. クロスツールチェーン

演習では CentOS(x86 アーキテクチャ)で、SUZAKU-V(PowerPC アーキテクチャ)用のアプリケーション開発を行います。アプリケーションが動作するアーキテクチャと、開発に使うコンピュータのアーキテクチャが異なる場合、クロスツールチェーンを構築する必要があります。今回は、以下のツールを使用してクロスツールチェーンを構築します。詳しくは 2 章で説明します。

binutils

GCC

Linux カーネルヘッダ

uClibc

2. SUZAKU-V の設定

演習では、アプリケーション開発のほか、Linux カーネル(v2.6.31)を導入します。しかし、出荷状態の SUZAKU-V では Linux カーネル(v2.6.31)を実行することができません。そのため、フラッシュメモリ内のブートローダ Hermit を書き換えて、Linux カーネル(v2.6.31)を実行できるようにします。詳しくは 3 章で説明します。

また、LED/SW ボードを使用するためには、フラッシュメモリの FPGA 初期化データを書き換える必要があります。詳しくは 4 章で説明します。

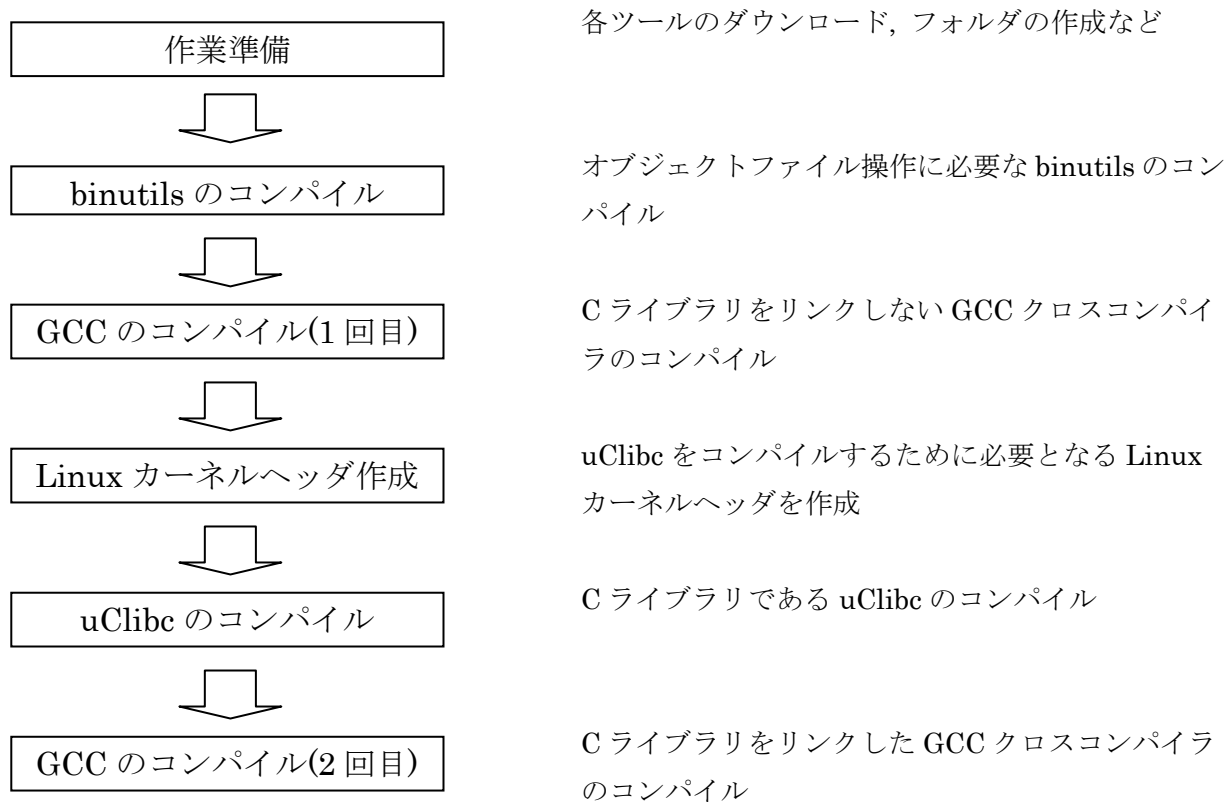
尚、これらの作業は初回のみ行います。以降、演習内容や内部回路に変更がない限り、これらの作業を行う必要はありません。

3. 演習前の準備

演習を行う前に、使用するディレクトリやファイルを準備する必要があります。詳しくは 5 章で説明します。また、この準備は、演習の前に毎回行う必要があります。

2章 クロスツールチェーンの構築

この章では, CentOS(x86 アーキテクチャ)上で, PowerPC 用のソフトウェアを作成するためのクロスツールチェーンを構築します. 尚, この章での作業は以下のような手順で行います.



1. ツールチェーンの構築に必要なもの

今回クロスツールチェーンの構築に使用した OS は CentOS 5.4 です. また, ツールチェーンの構築には, 以下のツールが必要です.

- binutils
 - GNU アセンブラ(as) やリンカ(ld) 等オブジェクトファイル操作に必要なバイナリユーティリティ
- GCC
 - c, c++, java など, 多数のプログラム言語のコンパイラコレクション
- uClibc
 - 組み込み Linux 向けの小型標準 C ライブラリ
- Linux カーネルヘッダ
 - Linux カーネルのヘッダファイル

GCC のコンパイルに必要なパッケージ

- gmp
 - 整数, 有理数, 浮動小数点数に対応して, 基本的な四則演算が行える任意精度数演算ライブラリ
- mpfr
 - 浮動小数点数による多倍長演算ライブラリ

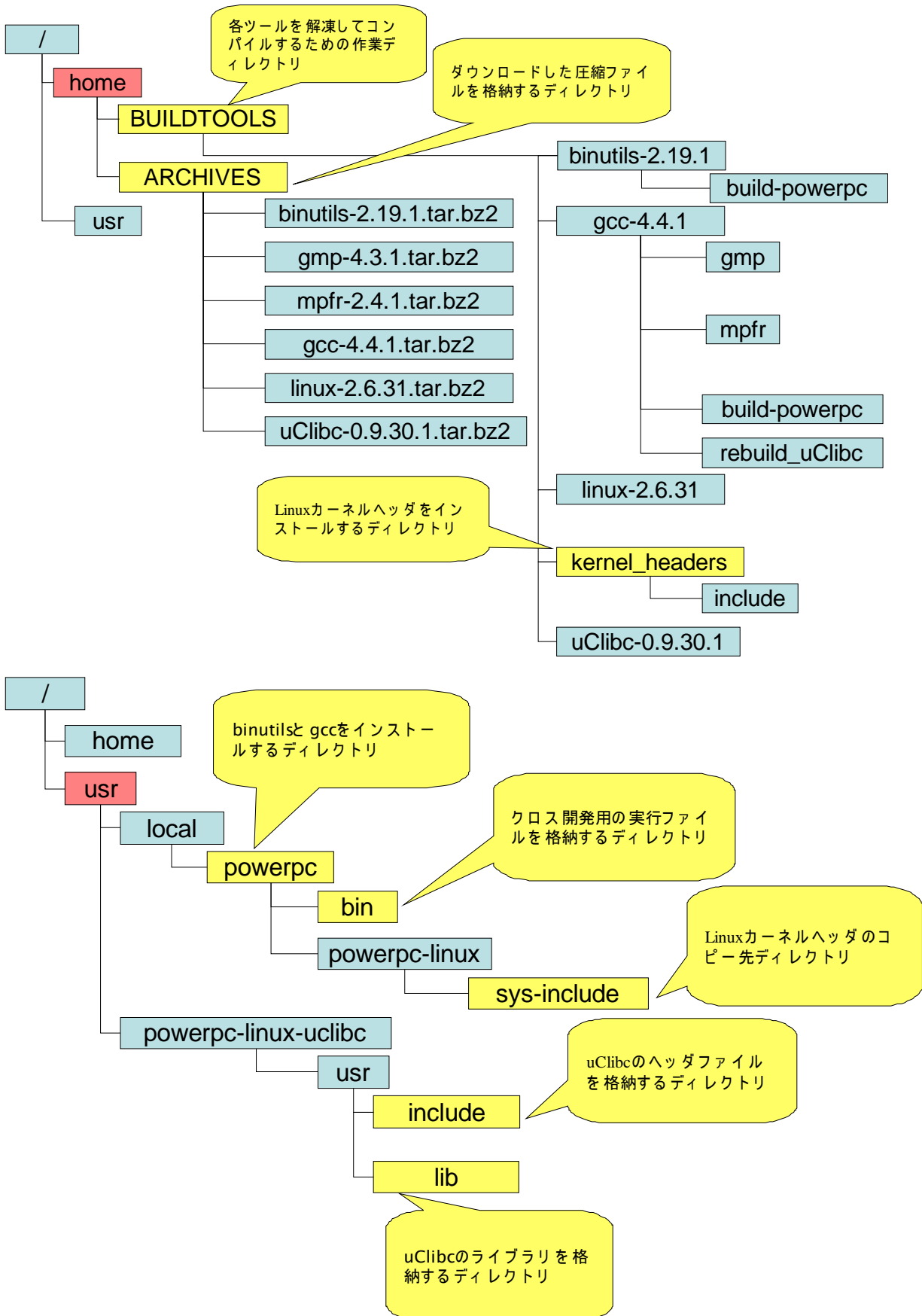
今回は下記のツールを使用します. 各ツールの tar.bz2 形式ファイルをダウンロードしてください.

パッケージ	バージョン	配布元
binutils	2.19.1	http://ftp.gnu.org/gnu/binutils/
GCC	4.4.1	http://ftp.gnu.org/gnu/gcc/
uClibc	0.9.30.1	http://www.uclibc.org/downloads/
Linux	2.6.31	http://www.kernel.org/pub/linux/kernel/v2.6/
gmp	4.3.1	http://ftp.gnu.org/gnu/gmp/ *1
mpfr	2.4.1	http://ftp.gnu.org/gnu/mpfr/ *1

*1: GCC のコンパイルに必要なパッケージ

2. 作業準備

クロスツールチェーンを構築するためのディレクトリ構成は以下のようになっています。



ディレクトリの定義

- ARCHIVES … ダウンロードした圧縮ファイルを入れるディレクトリ
- BUILDTOOLS … 各ツールを解凍してコンパイルするための作業ディレクトリ
- kernel_headers … Linux カーネルヘッダをインストールするディレクトリ
- powerpc/bin … クロス開発用の実行ファイルを格納するディレクトリ
- powerpc-include/sys-include … Linux カーネルヘッダのコピー先ディレクトリ
- powerpc-linux-uclibc … uClibc のライブラリとヘッダファイルを格納するディレクトリ

以下のコマンドで, home の下に上記のディレクトリを作成します. 以降はルートで作業を進めます.

```
sudo su
mkdir /home/ARCHIVES
mkdir /home/BUILDTOOLS
```

ダウンロードした各ツールの圧縮ファイルを ARCHIVES に入れてください.

仮想コンピュータ上で作業を行う場合, WinSCP 等でファイルを転送してください.

3. CentOS

CentOS に最低限の機能しかインストールしていない場合は GCC と ncurses-devel をインストールする必要があります. 以下のように GCC と ncurses-devel をインストールしてください.

```
yum install gcc
yum install ncurses-devel *1
```

*1: 以降の手順において, menuconfig 画面を表示するために必要となるライブラリ

4. binutils のインストール

オブジェクトファイルを取り扱うためのプログラミングツールとして, binutils をインストールします. binutils は以下のツールを含んでいます.

- as … アセンブラ
- ld … リンカ
- addr2line … プログラム内のアドレスをファイル名と行番号に変換する
- ar … アーカイブの作成、変更、および展開
- c++filt … C++シンボルのデマングルを行う
- nm … オブジェクトファイルに含まれるシンボル(クラス、関数など)を表示する
- objcopy … オブジェクトファイルをコピーする、オブジェクトフォーマットの変換を行う

- `objdump` … オブジェクトファイルのダンプ情報を表示する
- `ranlib` … アーカイブのインデックスを作成する
- `readelf` … ELF ファイルの中身を表示する
- `size` … セクションの大きさとその合計をリストする
- `strings` … 表示可能な文字列をリストする
- `strip` … オブジェクトファイル中のシンボルを除去する
- `gprof` … コール・グラフ (call graph) のプロファイルを表示する

以下のコマンドを実行し, `binutils` をインストールします.

```
cd /home/BUILDTOOLS
tar jxvf ../ARCHIVES/binutils-2.19.1.tar.bz2 *1
cd binutils-2.19.1
mkdir build-powerpc *2
cd build-powerpc
../configure --target=powerpc-linux --prefix=/usr/local/powerpc *3
make
make install
```

*1: `tar` コマンドで圧縮ファイルを解凍する

*2: コンパイルはソースツリーの外側で行うことが推奨されている(GCC も同様)

*3: “`-target=`” オプションでターゲットアーキテクチャを指定

“`-prefix=`” オプションでインストール先ディレクトリを指定

正常にインストールが完了すると, `/home/usr/local/powerpc/bin` に以下の実行ファイルが生成されます.

<code>powerpc-linux-addr2line</code>	<code>powerpc-linux-gprof</code>	<code>powerpc-linux-ranlib</code>
<code>powerpc-linux-ar</code>	<code>powerpc-linux-ld</code>	<code>powerpc-linux-readelf</code>
<code>powerpc-linux-as</code>	<code>powerpc-linux-nm</code>	<code>powerpc-linux-size</code>
<code>powerpc-linux-c++filt</code>	<code>powerpc-linux-objcopy</code>	<code>powerpc-linux-strings</code>
<code>powerpc-linux-embedspu</code>	<code>powerpc-linux-objdump</code>	<code>powerpc-linux-strip</code>

5. GCC のコンパイル(1 回目)

クロス開発用の標準 C ライブラリ (`uClibc`) をクロスコンパイルするために, 先に GCC (クロスコンパイラ) をインストールします.

ライブラリを含まない GCC のインストールは以下のように行います。

```
cd /home/BUILDTOOLS
tar jxvf ../ARCHIVES/gcc-4.4.1.tar.bz2
cd gcc-4.4.1
tar jxvf ../../ARCHIVES/gmp-4.3.1.tar.bz2
tar jxvf ../../ARCHIVES/mpfr-2.4.1.tar.bz2
mv gmp-4.3.1/ gmp
mv mpfr-2.4.1/ mpfr
mkdir build-powerpc
cd build-powerpc
../configure --target=powerpc-linux --prefix=/usr/local/powerpc --enable-languages=c *1
make all-gcc
make install-gcc
```

*1: “-enable-languages=c” で C 言語のコンパイラのみをインストールするように指定

正常にインストールが完了すると、/usr/local/powerpc/bin に以下の実行ファイルが生成されます。

```
powerpc-linux-gcc-4.4.1    powerpc-linux-gcc
```

6. Linux カーネルヘッダの作成

uClibc のライブラリをインストールするためには、Linux システムコールの情報が必要となるため、先に Linux カーネルヘッダをインストールします。Linux カーネルヘッダを以下のようにしてインストールします。

```
cd /home/BUILDTOOLS
tar jxvf ../ARCHIVES/linux-2.6.31.tar.bz2
cd linux-2.6.31
make mrproper *1
make ARCH=powerpc headers_check *2
make ARCH=powerpc INSTALL_HDR_PATH=/home/BUILDTOOLS/kernel_headers
headers_install *3
```

*1: 環境の初期化

*2: インストールに必要なヘッダファイルをチェックする

*3: “INSTALL_HDR_PATH=” で、ヘッダファイルのインストール先を指定

下記のディレクトリにヘッダファイルがインストールされます。

```
/home/BUILDTOOLS/kernel_headers
```

7. uClibc のクロスコンパイル

作成した GCC(クロスコンパイラ)と Linux カーネルヘッダを使用して、uClibc をクロスコンパイルします。

以下のようにして、ソースファイルを展開し、menuconfig を実行します。

```
cd /home/BUILDTOOLS
tar jxvf ../ARCHIVES/uClibc-0.9.30.1.tar.bz2
cd uClibc-0.9.30.1
PATH=$PATH:/usr/local/powerpc/bin *1
export PATH
make CROSS=powerpc-linux menuconfig *2
```

*1: /usr/local/powerpc/bin にパスを通す

*2: 'CROSS=' でクロス環境を指定

menuconfig 画面での操作方法は以下のとおりです。詳細は menuconfig 画面の上部に記載されています。

キー	操作
Enter	現在の項目を選択
delete	文字の削除
方向キー	カーソルの移動
Y	*印を入れる
N	*印をはずす

menuconfig を使用して、各項目を以下のように設定します。

```
Target Architecture (alpha) → Target Architecture (powerpc)
                               ↑ powerpc を選択
```

```
Target Architecture Features and Options →
Linux kernel header location (/home/BUILDTOOLS/kernel_headers/include)
                               ↑ ()内に作成したカーネルヘッダのパスを入力
```

General Library Settings →

- Generate only Position Independent Code (PIC) ← *印をはずす(N キー)
- Enable support for shared libraries ← *印をはずす(N キー)

Target Architecture Features and Options →

- Enable C99 Floating-point environment ← *印を入れる(Y キー)

exit を選択して設定を保存します。

以下のようにして uClibc をコンパイルしてください。

```
make CROSS=powerpc-linux-  
make CROSS=powerpc-linux- install
```

/usr/powerpc-linux/uclibc/usr/にヘッダファイルとライブラリがインストールされます。

```
include lib
```

8. GCC のコンパイル(2 回目)

Linux アプリケーションを作成するために、uClibc のライブラリをリンクした GCC(クロスコンパイラ)をインストールします。

uClibc をリンクした GCC のコンパイル時には、

“/usr/local/powerpc/powerpc-linux/sys-include”

内のヘッダファイルを参照するため、ディレクトリを作成し、Linux カーネルヘッダをコピーする必要があります。

```
mkdir /usr/local/powerpc/powerpc-linux/sys-include  
cp -r /home/BUILDTOOLS/kernel_headers/include/* ¥  
/usr/local/powerpc/powerpc-linux/sys-include/
```

1 回目とは異なるディレクトリで作業します。

```
cd /home/BUILDTOOLS/gcc-4.4.1  
mkdir rebuild_uClibc  
cd rebuild_uClibc
```

uClibc ライブラリをリンクした GCC をインストールします.

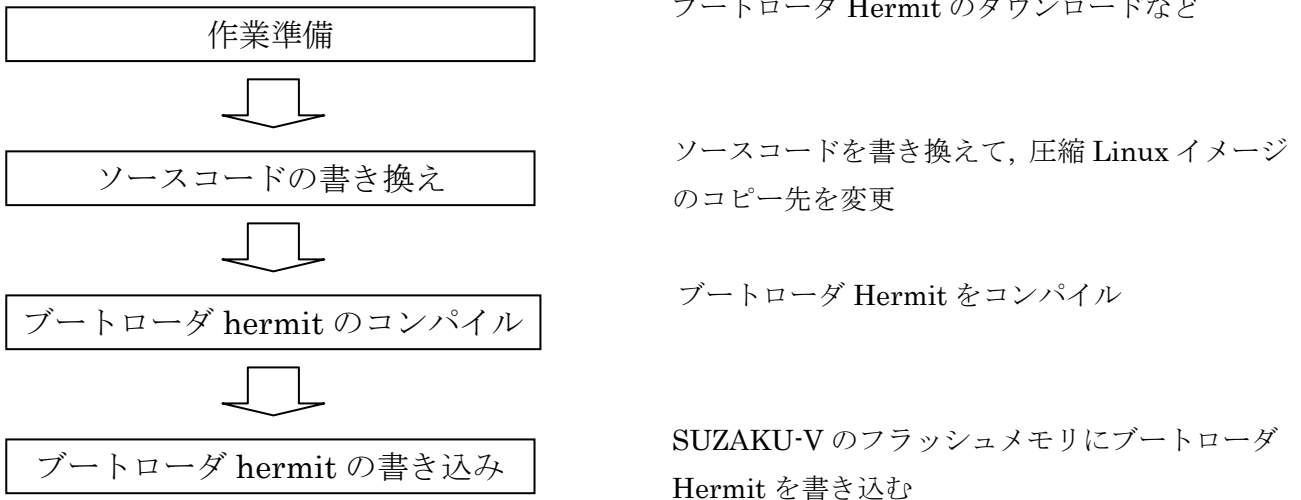
```
./configure --target=powerpc-linux --prefix=/usr/local/powerpc  ¥  
--with-headers=/home/BUILDTOOLS/kernel_headers/include/  ¥  
--with-headers=/usr/powerpc-linux-uclibc/usr/include/  ¥  
--with-libs=/usr/powerpc-linux-uclibc/usr/lib/ --disable-libmudflap  ¥  
--disable-libssp --disable-shared --disable-threads --disable-libgomp  ¥  
--enable-languages=c --disable-multilib *1  
make  
make install  
exit
```

- *1: "--with-headers=" で使用するヘッダを指定
"--with-libs=" で使用するライブラリを指定

これで, PowerPC 用のクロスツールチェーンが完成しました.

3章 ブートローダ Hermit の書き換え

ブートローダ Hermit の書き換えを行います。尚、この章での作業は以下のような手順で行います。



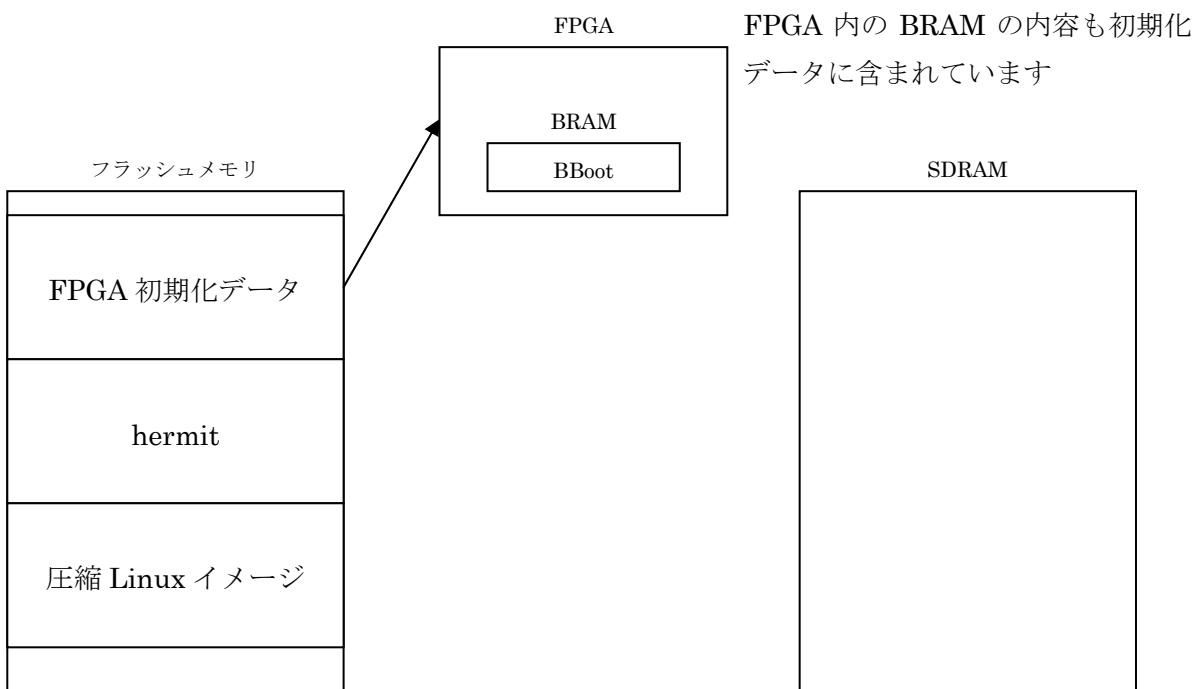
まずは、作業を行う前に、SUZAKU-V(出荷状態)のブートシーケンスを説明します。

1. SUZAKU-V(出荷状態)のブートシーケンス

SUZAKU-V(出荷状態)のブートシーケンスは以下の手順で行われます。

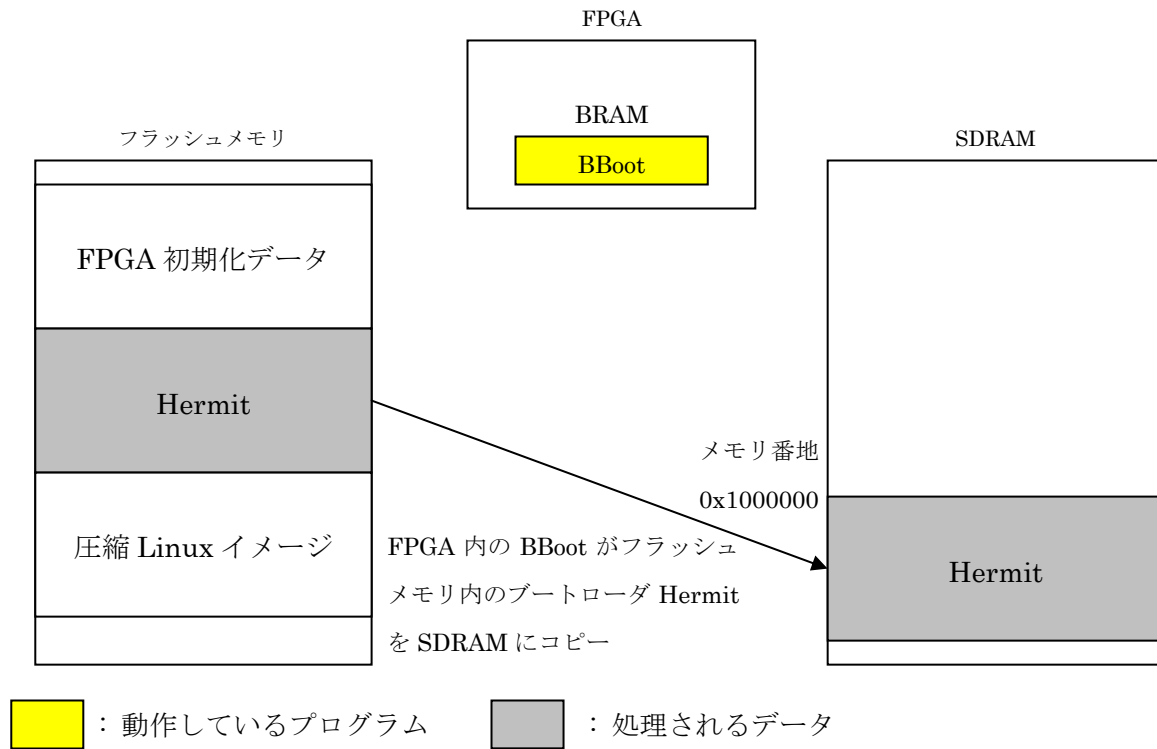
① FPGA コンフィギュレーション

SUZAKU-V に電源を投入すると、フラッシュメモリ内のFPGA初期化データを使用してコンフィギュレーション(初期化)が行われます。



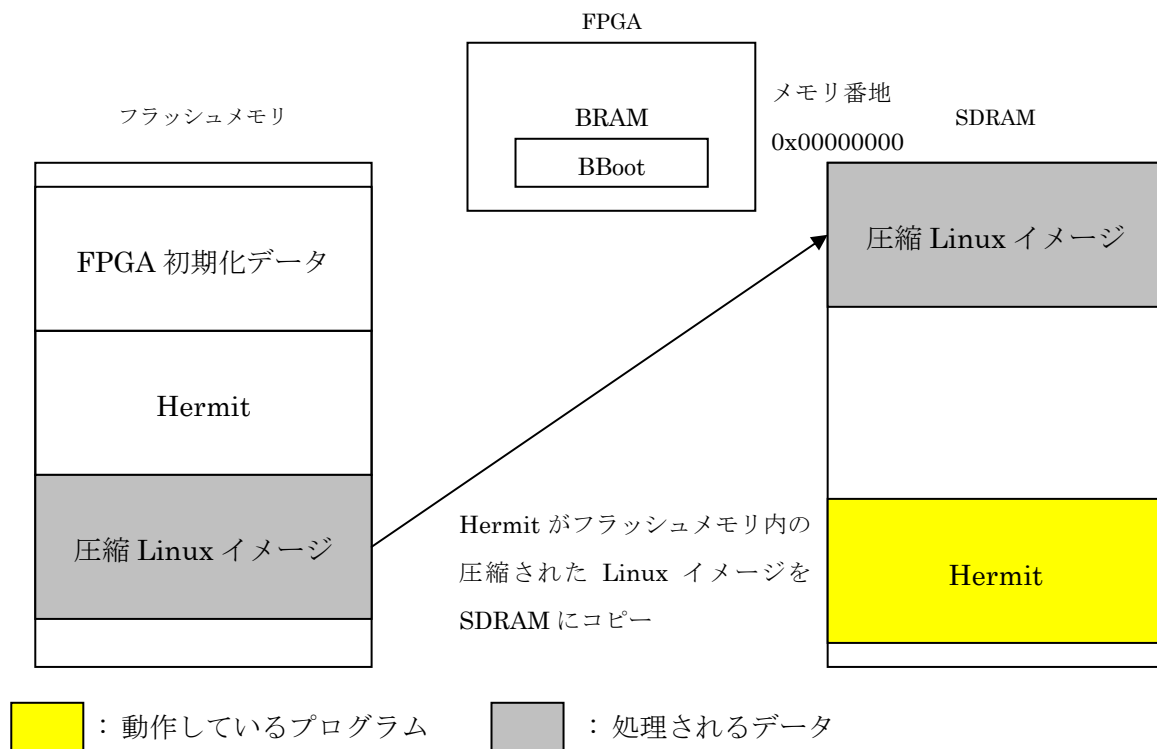
② BBootの起動

コンフィギュレーション終了後, FPGAのBRAM 内のBBootが起動し, ブートローダHermitを SDRAMへコピーします.



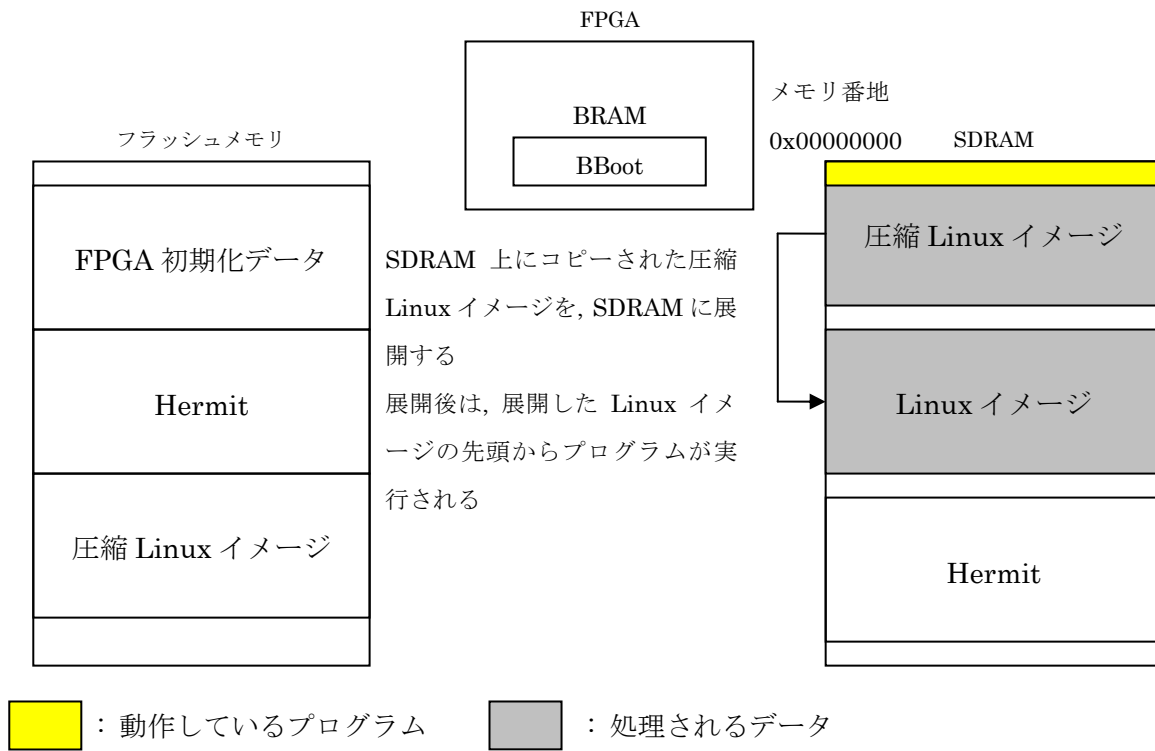
③ ブートローダHermitの起動

ブートローダHermit が起動し, 圧縮LinuxイメージをSDRAMにコピーします.



④ 圧縮Linuxイメージの展開

コピー終了後は圧縮Linuxイメージ内の展開プログラムが起動し、Linuxイメージが展開されます。



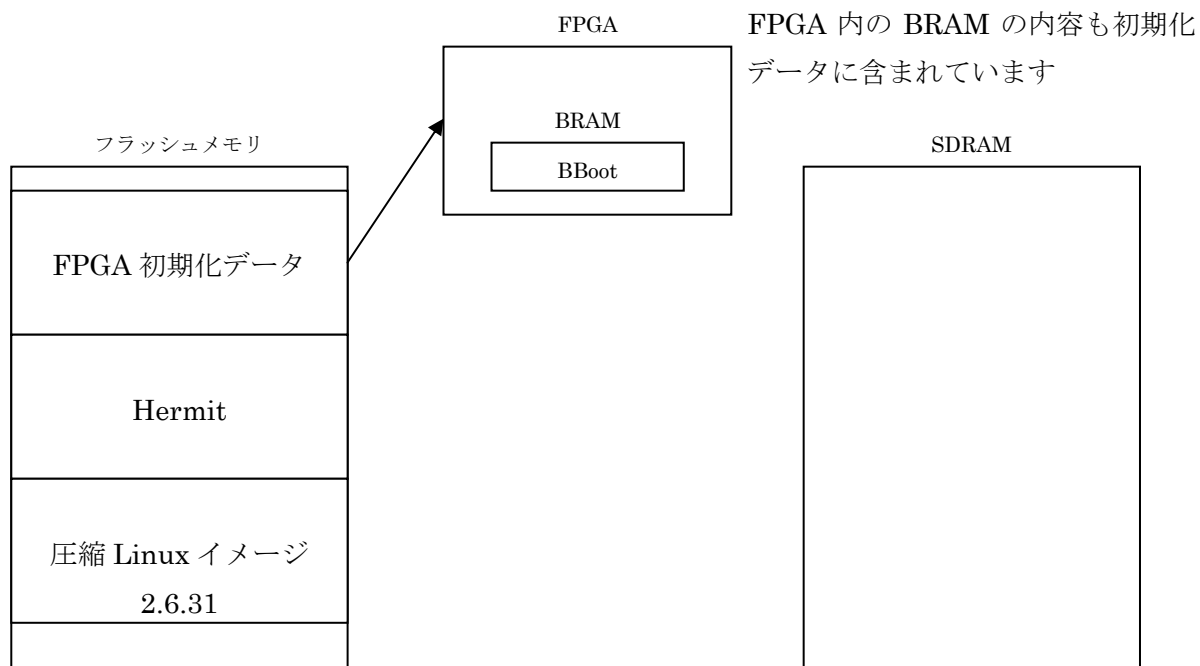
2. ブートローダ Hermit の書き換え

演習では, Linux-2.6.31 を構築し, 圧縮 Linux イメージを書き換えます.

Linux-2.6.31 を標準的に構築すると, 出荷状態のブートシーケンスに従い Linux イメージが展開されます. しかし, 展開された Linux イメージは 0x00000000 を先頭に配置されることを前提としているので, このままではプログラムを実行することができません. そのため, 圧縮 Linux イメージのコピー先アドレスを変更し, 以下のようなブートシーケンスになるようにブートローダ Hermit を書き換える必要があります. 通常, この作業は初回のみ行います. 以降, 演習内容に変更がない限りは, この作業を行う必要はありません.

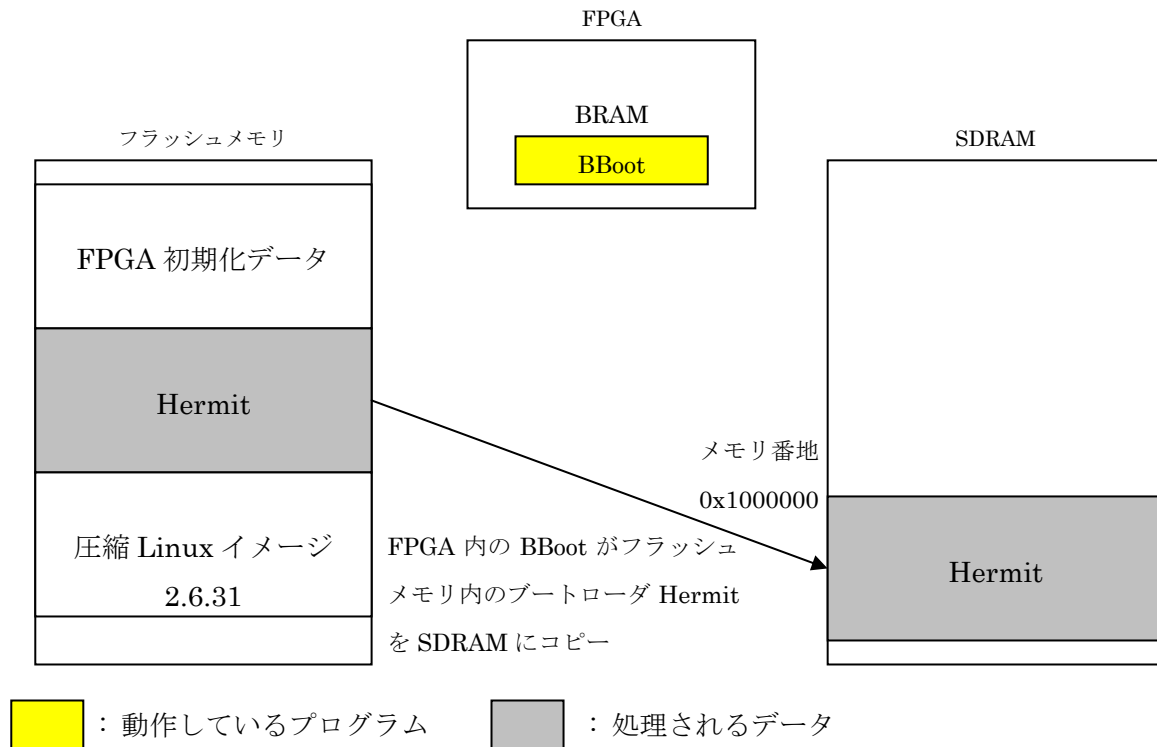
① FPGA コンフィギュレーション

SUZAKU-V に電源を投入すると, フラッシュメモリ内のFPGA初期化データを使用してコンフィギュレーション(初期化)が行われます.



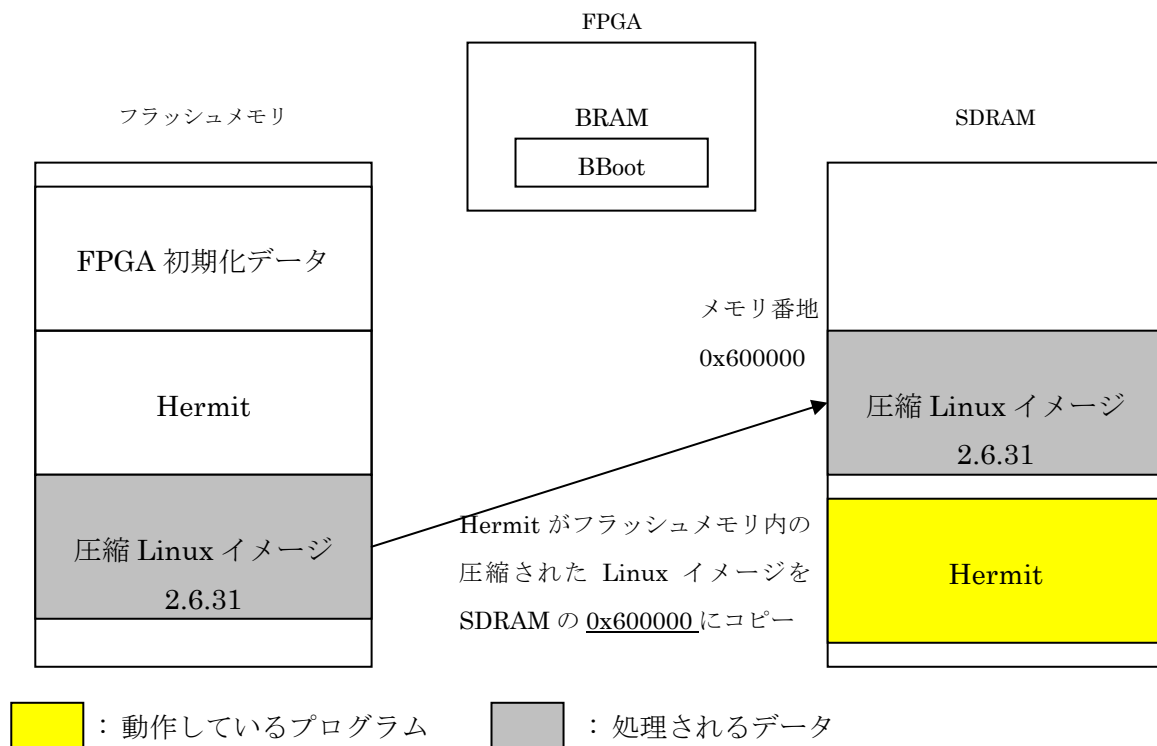
② BBootの起動

コンフィギュレーション終了後, FPGAのBRAM 内のBBootが起動し, ブートローダHermitをSDRAMへコピーします.



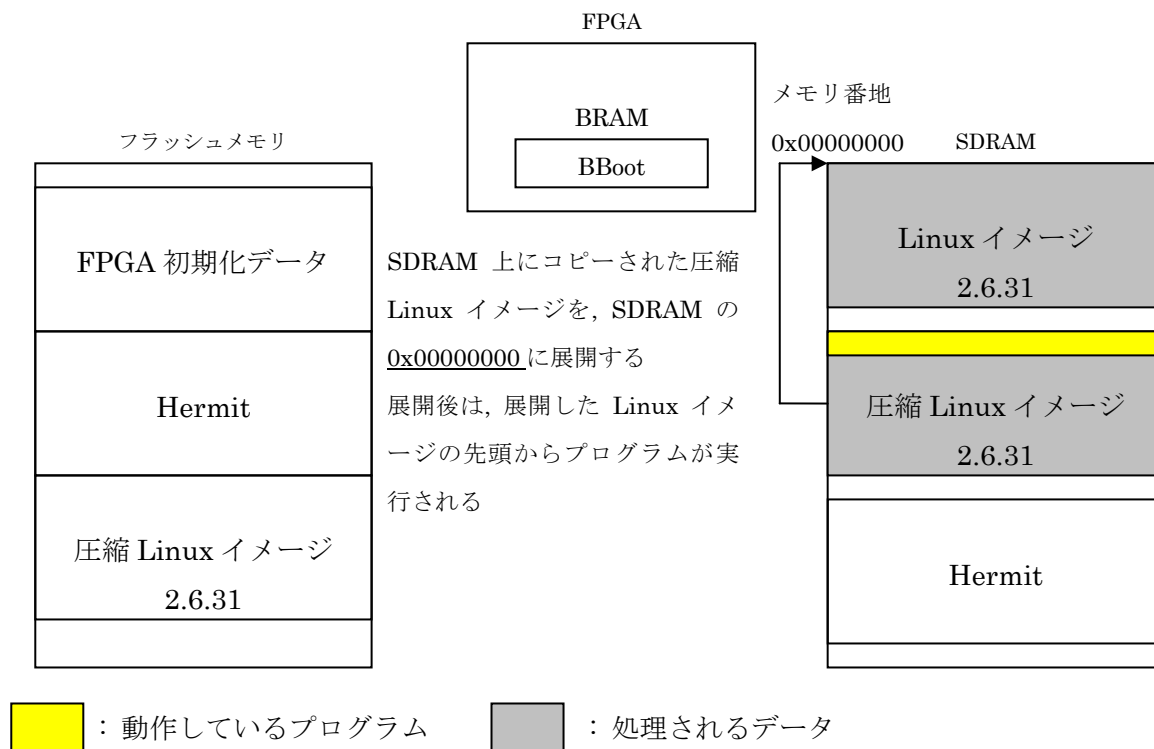
③ ブートローダHermitの起動

ブートローダHermit が起動し, 圧縮LinuxイメージをSDRAMにコピーします.



④ 圧縮Linuxイメージの展開

コピー終了後は圧縮Linuxイメージ内の展開プログラムが起動し、Linuxイメージが展開されます。



以降では、手順③のように、圧縮 Linux イメージが SDRAM の 0x00600000 にコピーされるようブートローダ Hermit を書き換えます。

3. 準備作業

ブートローダ Hermit をダウンロードします。下記の配布元から "hermit-at-1.1.21.tar.gz" をダウンロードしてください。

パッケージ	バージョン	配布元
hermit-at	1.1.21	http://suzaku.atmark-techno.com/filebrowser/tools

以降の作業はルートで行ってください。

ダウンロードしたファイルを ARCHIVES ディレクトリに格納してください。

以下のコマンドで home ディレクトリに Hermit を展開します。

```
cd /home
tar zxvf /home/ARCHIVES/hermit-at-1.1.21.tar.gz
```

4. ブートローダ Hermit のコンパイル

圧縮 Linux イメージのコピー先アドレスを変更して、コンパイルします。

テキストエディタで `src/target/suzaku/`内の `linux.c` を変更します。

```
cd hermit-at-1.1.21
vi src/target/suzaku/linux.c
```

`linux.c` の 80 行目を以下のように変更します。

変更前

```
kernel = (kernel_t)0;
```

変更後

```
kernel = (kernel_t)0x00600000;
```

赤字は変更箇所

`src/target/suzaku/`内の `memregions.h` を変更します

```
vi src/target/suzaku/memregions.h
```

47 行目を以下のように変更します。

変更前

```
#define LINUX_LOAD_ADDRESS    (DRAM_BASEADDR)
```

変更後

```
#define LINUX_LOAD_ADDRESS    (DRAM_BASEADDR + 0x00600000)
```

以下のコマンドで Hermit をコンパイルします。

```
PATH=$PATH:/usr/local/powerpc/bin *1
export PATH *1
make TARGET=suzaku PROFILE=powerpc *2
```

*1: すでに `PATH` が通っていれば不要

*2: “`TARGET=`” でターゲットボードを指定

“`PROFILE=`” でターゲットアーキテクチャを指定

コンパイルが完了すると `hermit-at_1.1.21/src/target/suzaku` 内に

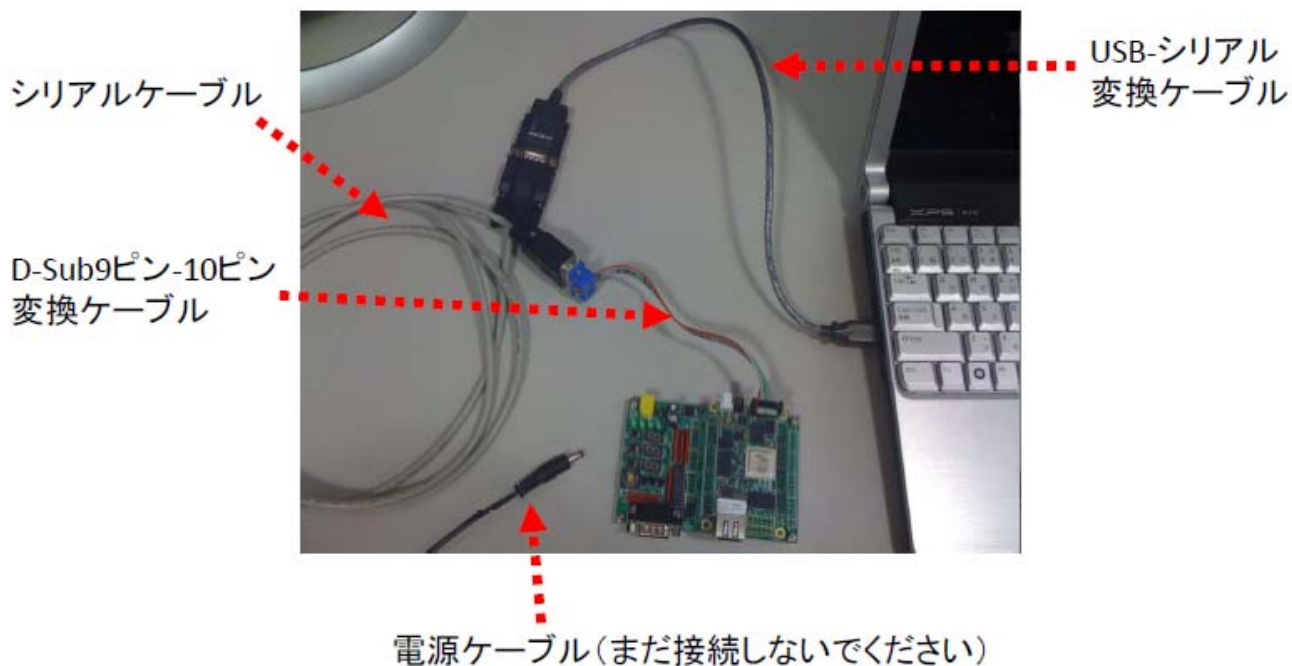
`loader-suzaku-powerpc-v1.1.21-8M.srec`

が生成されます。このファイルを WinSCP 等で PC に転送してください。

5. ブートローダ Hermit の書き込み

生成された `srec` 形式ファイルを SUZAKU-V に書き込みます。USB デバイスドライバがインストールされていない場合は、付属の CD-ROM からインストールしてください。

図のようにして、SUZAKU-V と PC を接続します。



Tera Term を起動し、シリアルポートに接続してください。



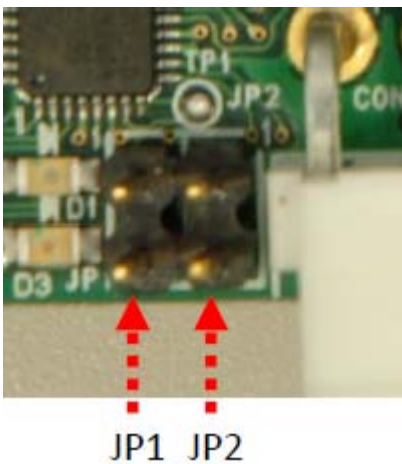
[設定]→[シリアルポート]からボー・レートを 115200 に設定します.

ポート(P):	COM9	OK
ボー・レート(B):	115200	
データ(D):	8 bit	キャンセル
パリティ(A):	none	
ストップ(S):	1 bit	ヘルプ(H)
フロー制御(F):	none	

送信遅延

0	ミリ秒/字(C)	0	ミリ秒/行(L)
---	----------	---	----------

電源を入れる前に, SUZAKU-V の JP1 にジャンパソケットを差し込みます.



ジャンパソケット

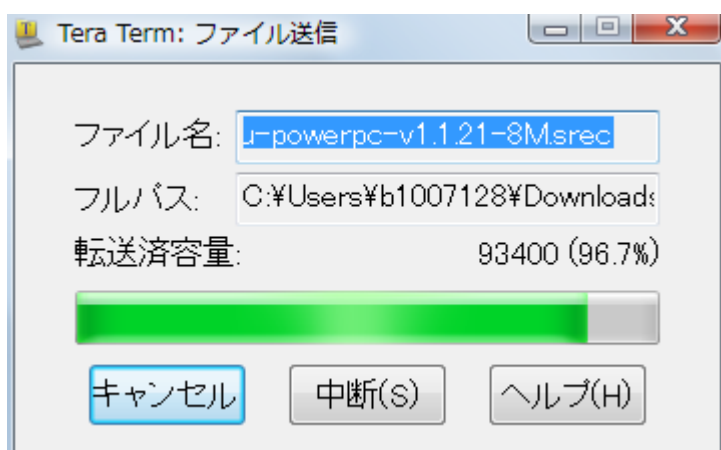


SUZAKU-V の電源を入れます. 起動したら Z キーを長押ししてください. “Please choose one of the following and hit enter.” と出たら S キーを押してください.

```
BBoot v2.10 (powerpc) compiled at 22:09:07, Apr 24 2009
Press 'z' or 'Z' for BBoot Menu.

Please choose one of the following and hit enter.
a: activate second stage bootloader (default)
s: download a s-record file
Start sending S-Record!!
```

[ファイル]→[ファイル送信]で loader-suzaku-powerpc-v1.1.21-8M.srec を SUZAKU-V に書き込みます。



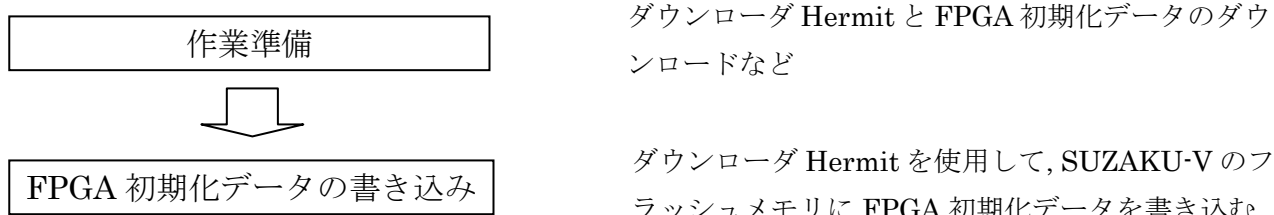
これでブートローダ **Hermit** の書き換えが完了しました。

4 章 FPGA 初期化データの書き換え

LED/SW ボードを使用するためには、フラッシュメモリの FPGA 初期化データを書き換える必要があります。通常、この作業は初回のみ行います。以降、内部回路を変更しない限りは、この作業を行う必要はありません。

フラッシュメモリへの書き込み方法はいくつかありますが、今回はダウンローダ **Hermit** での書き込み手順を説明します。そのほかの書き込み方法は **SUZAKU-V** スターターキットマニュアル(FPGA 開発編)(Linux 開発編)に記載されています。

尚、この章での作業は以下のような手順で行います。



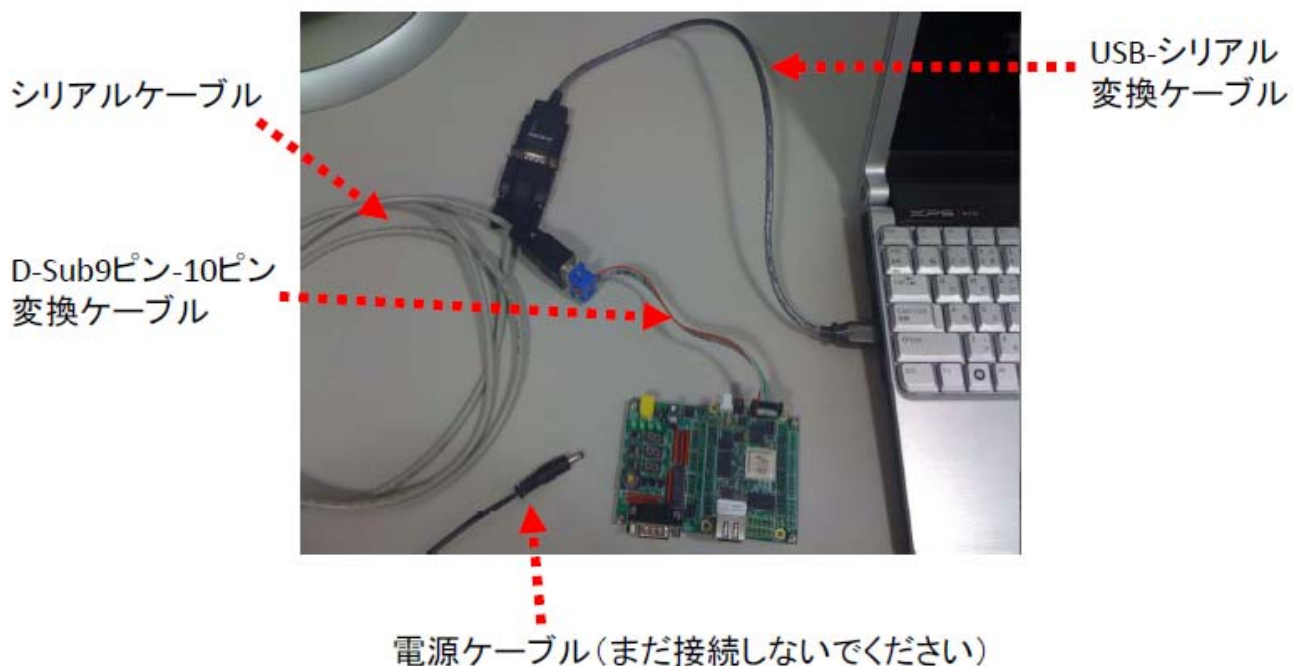
1. 作業準備

ダウンローダ **Hermit** と FPGA 初期化データをダウンロードします。下記の配布元から各ファイルをダウンロードしてください。

ファイル名	配布元
hermit-at-win_20090325.zip	http://suzaku.atmark-techno.com/filebrowser/tools
fpga-sz410-sil-gpio_control-101i-20090427.bin	http://suzaku.atmark-techno.com/filebrowser/suzaku-starter-kit/image

2. FPGA 初期化データの書き込み

次に, SUZAKU-V と PC を接続します.



Tera Term を起動し, シリアルポートに接続してください.

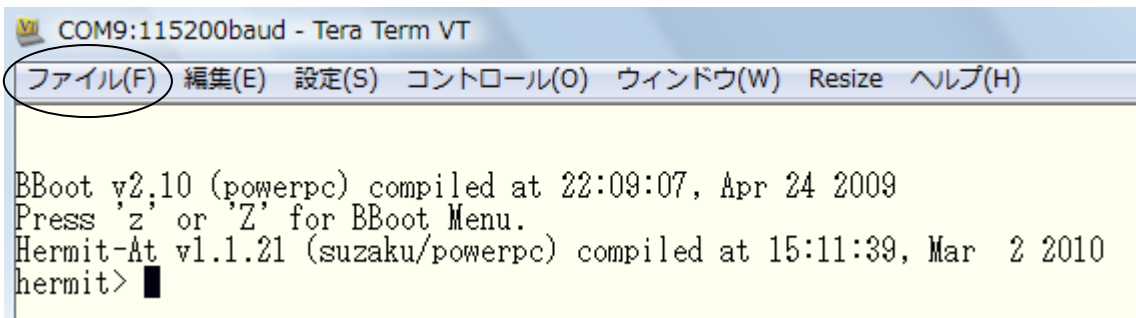


[設定]→[シリアルポート]からボー・レートを 115200 に設定します。

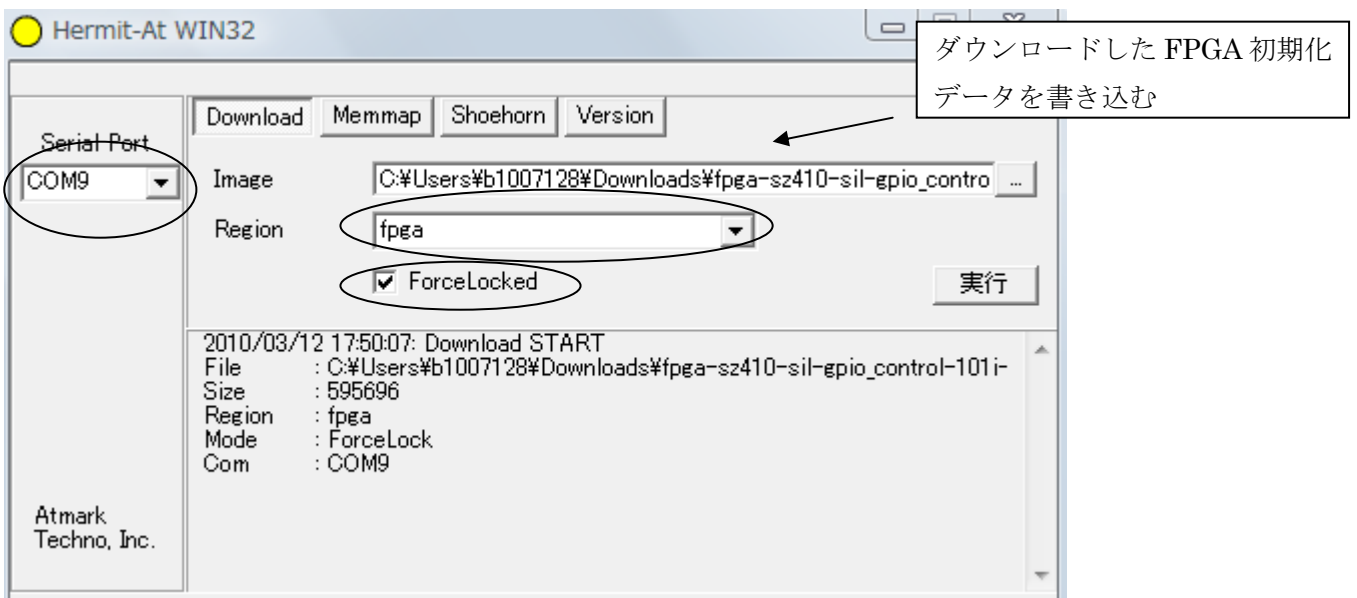
ポート(P): COM9
ボー・レート(B): 115200
データ(D): 8 bit
パリティ(A): none
ストップ(S): 1 bit
フロー制御(F): none

送信遅延
0 ミリ秒/字(C) 0 ミリ秒/行(L)

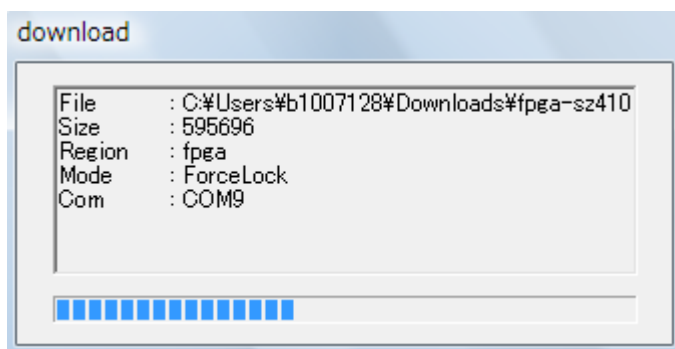
SUZAKU-V の JP1 にジャンパソケットを差し込み、SUZAKU-V の電源を入れます。ブートローダ Hermit が起動したら[ファイル]→[接続断]をします。



Hermit-at-win を起動し、以下のようにして FPGA 初期化データを書き込みます。



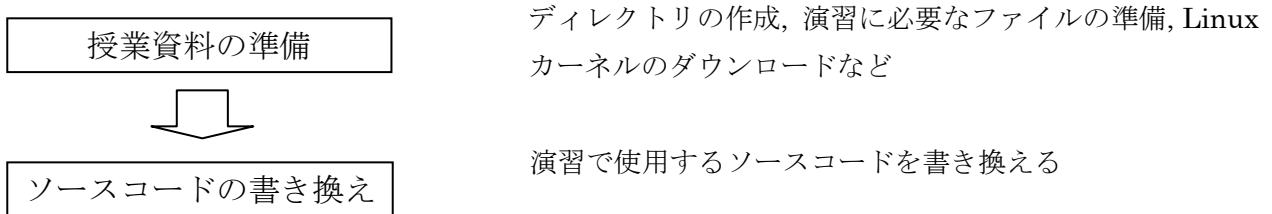
実行すると、書き込みが始まります。



これで、FPGA 初期化データの書き換えができました。

5章 授業資料の準備

演習を行うにあたって必要なディレクトリやファイルを準備します。作業はルートで行ってください。尚、この章での作業は以下のような手順で行います。



1. 演習第 13 回

ディレクトリを以下のように定義します。

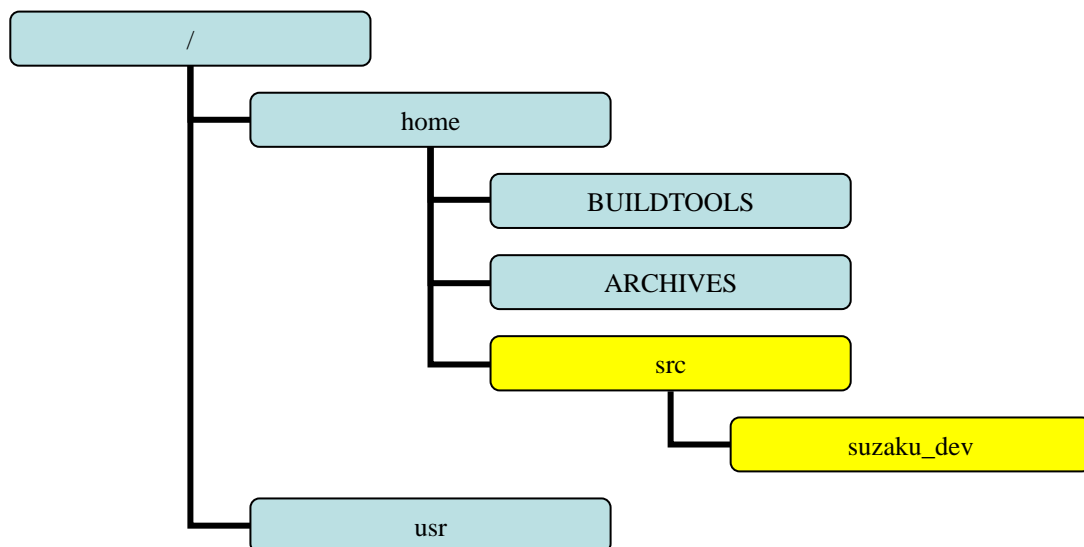
`src` … 演習に必要なソースファイルを入れるディレクトリ

`suzaku_dev` … SUZAKU-V のデバイスファイルを入れるディレクトリ

以下の手順で授業資料の準備を行います。

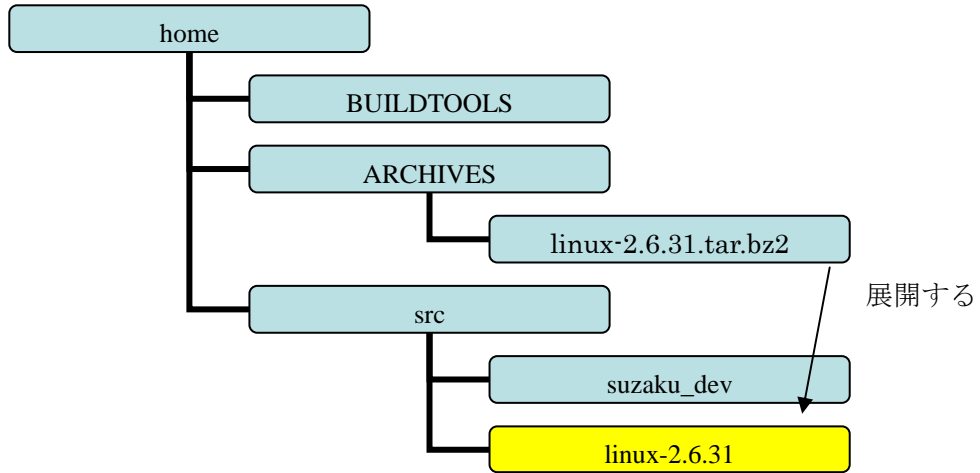
- ① `home` の下に `src` ディレクトリ, `src` の下に `suzaku_dev` ディレクトリを作成します。

```
mkdir /home/src
mkdir /home/src/suzaku_dev
```

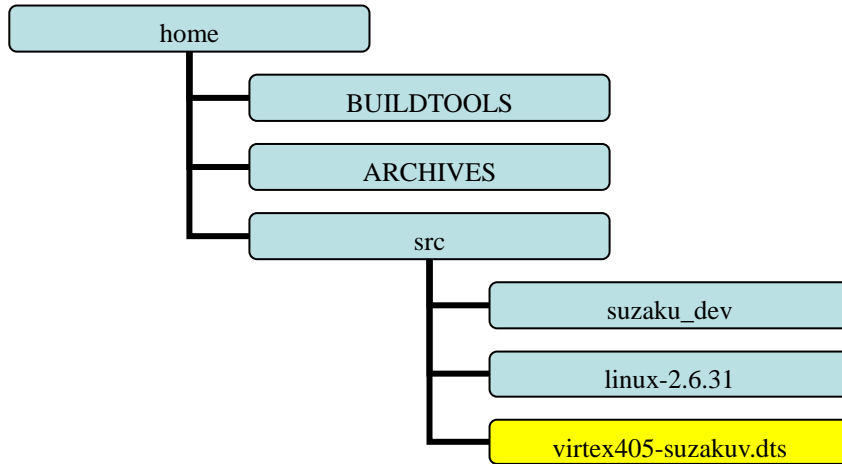


② ARCHIVES 内の linux-2.6.31.tar.bz2 を/home/src へ展開します。

```
cd /home/src  
tar jxvf ../ARCHIVES/linux-2.6.31.tar.bz2
```



③ デバイスツリーファイル virtex405-suzakuv.dts を/home/src へコピーします。



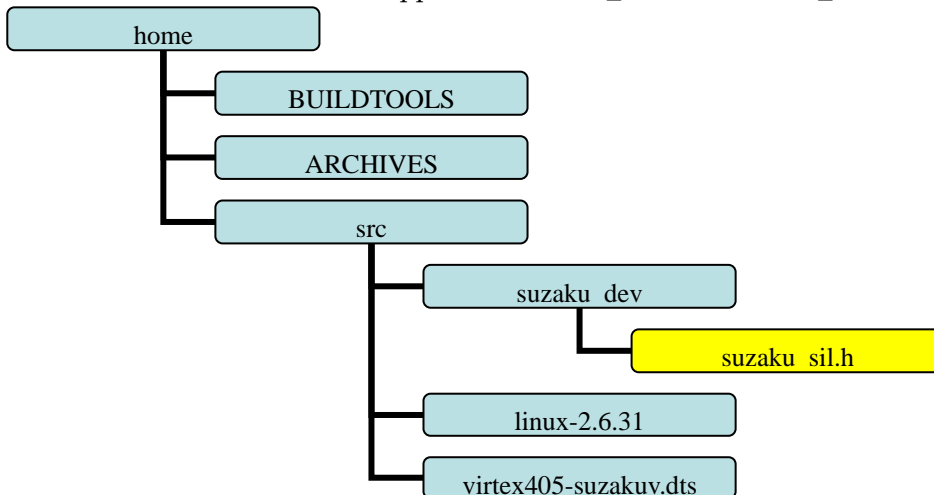
これで第 13 回の準備は終わりです。

2. 演習第 14 回

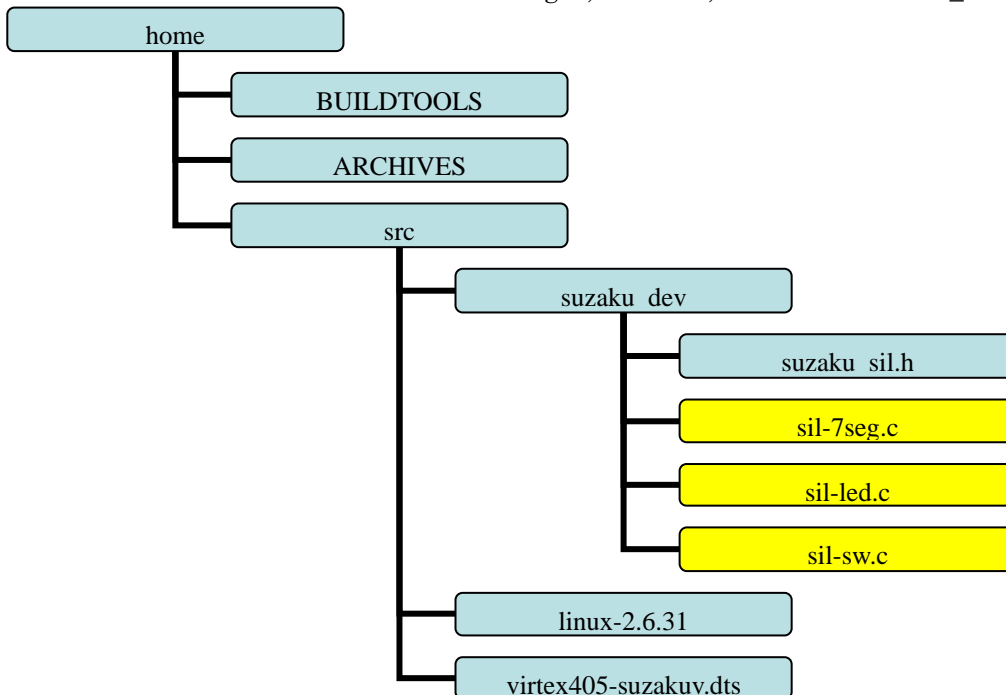
- ① 配布元から Linux カーネルをダウンロードし、展開してください。

ファイル名	配布元
linux-2.6.18-at11.tar.gz	http://suzaku.atmark-techno.com/filebrowser/dist

- ② linux-2.6.18-at11/include/asm-ppc 内の suzaku_sil.h を suzaku_dev にコピーします。



- ③ linux-2.6.18-at11/drivers/char 内の sil-7seg.c , sil-led.c , sil-sw.c を suzaku_dev にコピーします。



④ `suzaku_sil.h` のソースを変更します。

```
vi /home/src/suzaku_dev/suzaku_sil.h
```

下図のように `suzaku_sil.h` を変更してください。変更する箇所は以下のとおりです。

- 12 行目を削除する
- 14 行目-25 行目を変更する

	<code>suzaku_sil.h(変更前)</code>
12 行目	<code>#include <platforms/4xx/xparameters/xparameters.h></code>
13 行目	
14 行目	<code>#define SIL_7SEG_BASEADDR XPAR_SIL_7SEG_BASEADDR</code>
15 行目	<code>#define SIL_7SEG_HIGHADDR XPAR_SIL_7SEG_HIGHADDR</code>
16 行目	<code>#define SIL_7SEG_REMAP_SIZE (SIL_7SEG_HIGHADDR - SIL_7SEG_BASEADDR + 1)</code>
17 行目	<code>#define SIL_LED_BASEADDR XPAR_SIL_LED_BASEADDR</code>
18 行目	<code>#define SIL_LED_HIGHADDR XPAR_SIL_LED_HIGHADDR</code>
19 行目	<code>#define SIL_LED_REMAP_SIZE (SIL_LED_HIGHADDR - SIL_LED_BASEADDR + 1)</code>
20 行目	<code>#define SIL_SW_BASEADDR XPAR_SIL_SW_BASEADDR</code>
21 行目	<code>#define SIL_SW_HIGHADDR XPAR_SIL_SW_HIGHADDR</code>
22 行目	<code>#define SIL_SW_REMAP_SIZE (SIL_SW_HIGHADDR - SIL_SW_BASEADDR + 1)</code>
23 行目	<code>#define SIL_RSW_BASEADDR XPAR_SIL_RSW_BASEADDR</code>
24 行目	<code>#define SIL_RSW_HIGHADDR XPAR_SIL_RSW_HIGHADDR</code>
25 行目	<code>#define SIL_RSW_REMAP_SIZE (SIL_RSW_HIGHADDR - SIL_RSW_BASEADDR + 1)</code>

削除箇所



	<code>suzaku_sil.h(変更後)</code>
12 行目	
13 行目	
14 行目	<code>#define SIL_7SEG_BASEADDR 0xF0FFD000</code>
15 行目	<code>#define SIL_7SEG_HIGHADDR 0xF0FFD1FF</code>
16 行目	<code>#define SIL_7SEG_REMAP_SIZE (SIL_7SEG_HIGHADDR - SIL_7SEG_BASEADDR + 1)</code>
17 行目	<code>#define SIL_LED_BASEADDR 0xF0FFD200</code>
18 行目	<code>#define SIL_LED_HIGHADDR 0xF0FFD3FF</code>
19 行目	<code>#define SIL_LED_REMAP_SIZE (SIL_LED_HIGHADDR - SIL_LED_BASEADDR + 1)</code>
20 行目	<code>#define SIL_SW_BASEADDR 0xF0FFD400</code>
21 行目	<code>#define SIL_SW_HIGHADDR 0xF0FFD5FF</code>
22 行目	<code>#define SIL_SW_REMAP_SIZE (SIL_SW_HIGHADDR - SIL_SW_BASEADDR + 1)</code>
23 行目	<code>#define SIL_RSW_BASEADDR 0xF0FFD600</code>
24 行目	<code>#define SIL_RSW_HIGHADDR 0xF0FFD7FF</code>
25 行目	<code>#define SIL_RSW_REMAP_SIZE (SIL_RSW_HIGHADDR - SIL_RSW_BASEADDR + 1)</code>

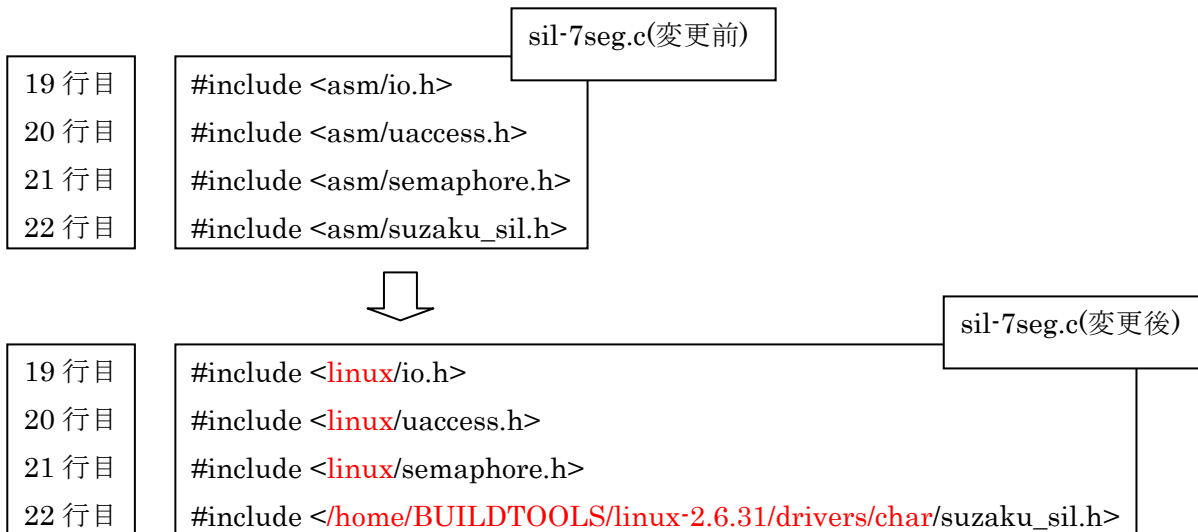
赤字は変更箇所です

⑤ sil-7seg.c のソースを変更します。

```
vi /home/src/suzaku_dev/sil-7seg.c
```

下図のように sil-7seg.c を変更してください。変更する箇所は以下のとおりです。

- ・ 19 行目-22 行目を変更する

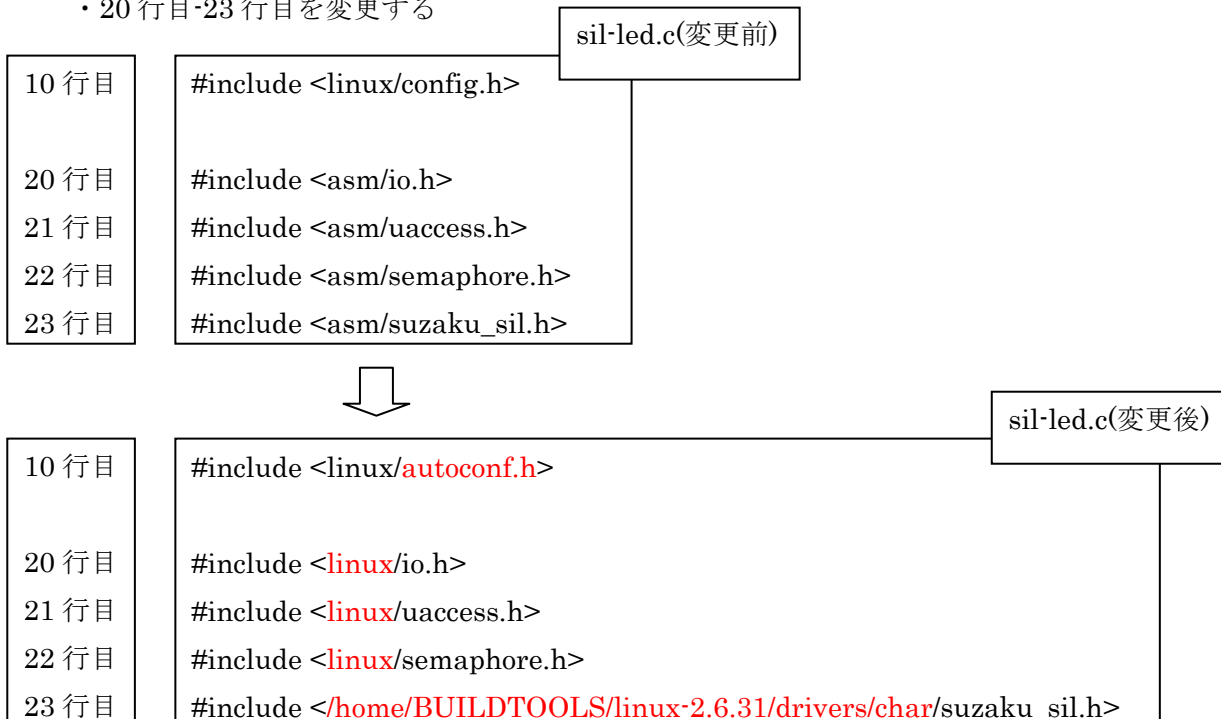


⑥ sil-led.c のソースを変更します。

```
vi /home/src/suzaku_dev/sil-led.c
```

下図のように sil-led.c を変更してください。変更する箇所は以下のとおりです。

- ・ 10 行目を変更する
- ・ 20 行目-23 行目を変更する

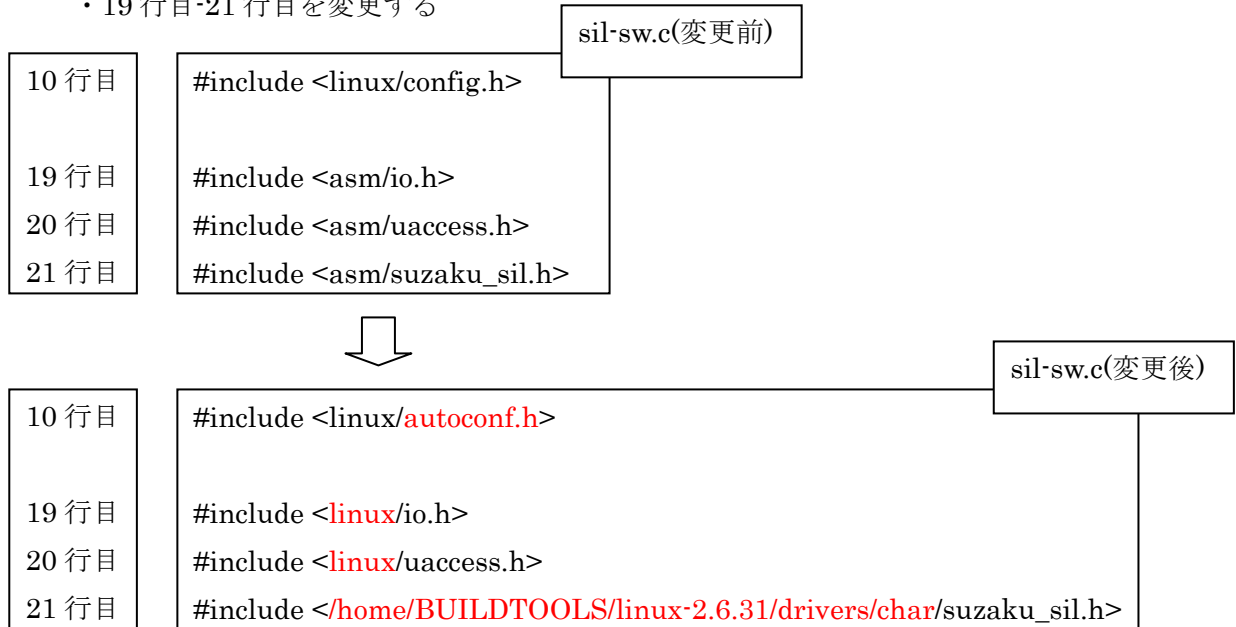


⑦ sil-sw.c のソースを変更します。

```
vi /home/src/suzaku_dev/sil-sw.c
```

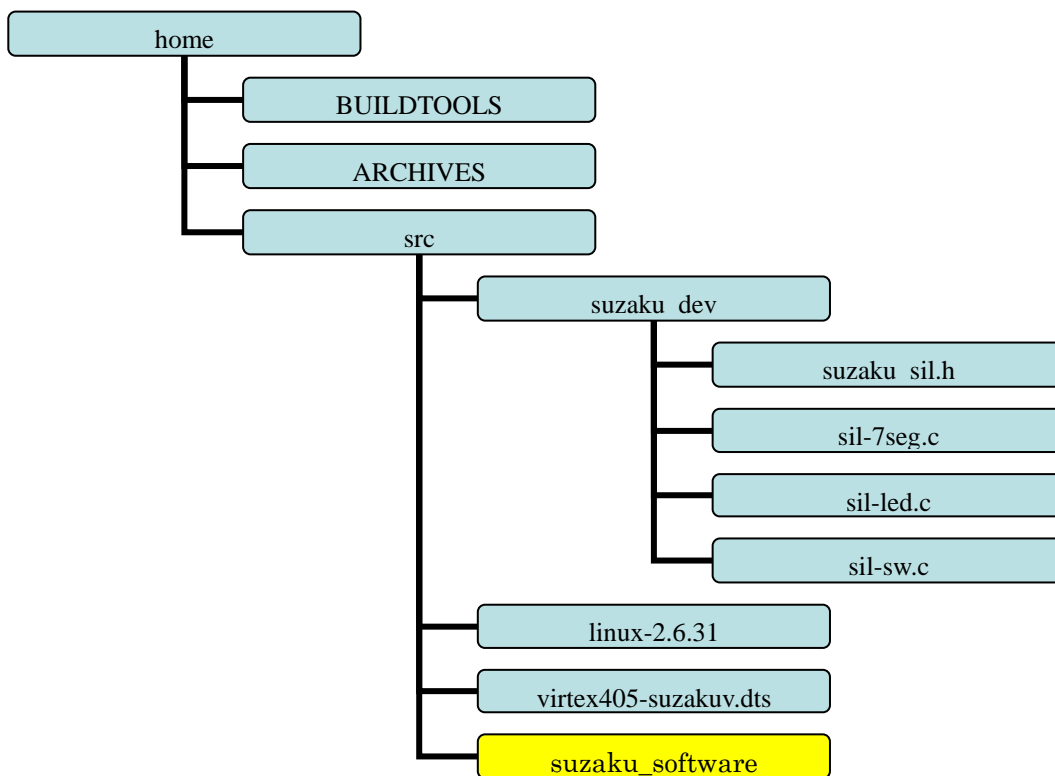
下図のように sil-sw.c を変更してください。変更する箇所は以下のとおりです。

- ・ 10 行目を変更する
- ・ 19 行目-21 行目を変更する

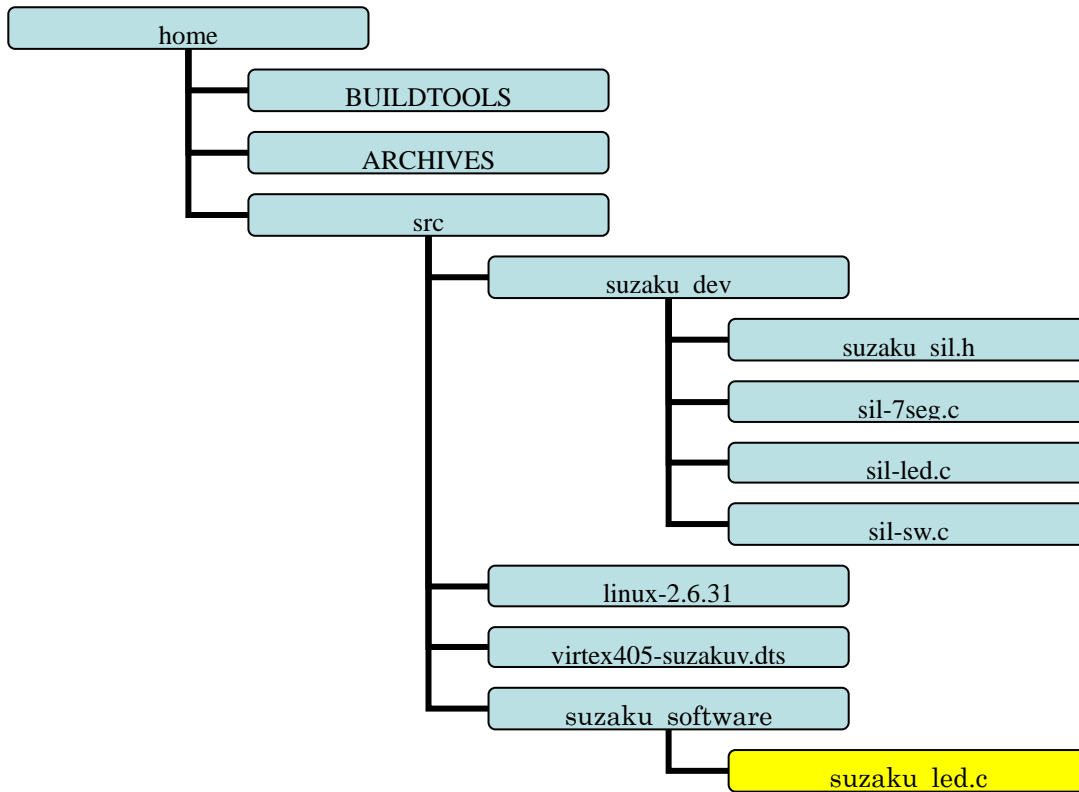


⑧ src の下に suzaku_software を作成します。

```
mkdir /home/src/suzaku_software
```



⑨ サンプルプログラム `suzaku_led.c` を `suzaku_software` にコピーします。



これで第 14 回の準備は終わりです。