FPGA内のハードウェア用にLinuxデバイス・ドライバを作成する

田中 一郎

近年，FPGAの使用により，独自のハードウェアを手軽に作成できるようになった．それらの独自ハードウェアをOS上で動作するアプリケーションから利用するにはデバイス・ドライバが必要になる．
本稿では，まずFPGA内に1,024バイトのメモリを作成する．そしてLinuxからアクセスするためのキャラクタ・デバイス・ドライバも同時に作成する．（編集部）

本稿では，アットマークテクノ社のSUZAKU-V（SZ310-U00）[注1]を例に，組み込みLinuxの実装を解説します．SUZAKU-VはFPGA内に独自の論理を作成できるため，開発者自身がハードウェアとソフトウェアを自由に組み込むことができます．従って，拡張性に富んだ柔軟なシステムを作成できます．FPGA内のCPUを使ったシステムを設計する際の入門機の用途としては手ごろなボードでしょう．

1．Linuxデバイス・ドライバ

さまざまなハードウェアの制御を行うに当たって，Linuxではアプリケーションから共通の手順でできる仕組みが提供されています．具体的には，まず「open」を使って制御を開始します．そして，「read」，「write」，「ioctl」を使ってハードウェアを制御し，「close」で終了します．Linuxカーネルでハードウェアを制御する部分がデバイス・ドライバです．

Linuxデバイス・ドライバには，大きく分けて，ブロック型とキャラクタ型の二つがあります．ブロック型は，カーネルと定められたバイト数単位でデータを転送します．キャラクタ型は転送単位が1バイトです．ここでは，キャラクタ型デバイスを例に解説します．

デバイス・ドライバはカーネルの一部として動作します．カーネルに静的リンク（スタティック・リンク）するか，起動後に動的リンク（ダイナミック・リンク）するかは，カーネルのコンパイル・オプションで決定します．

リスト1
file_operations 構造体

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long,
unsigned long);
#ifdef MAGIC_ROM_PTR
        int (*romptr) (struct file *, struct vm_area_struct *);
#endif /* MAGIC_ROM_PTR */
};
```

注1：CPUボード SZ310-U00，http://suzaku.atmark-techno.com/series/suzaku-v

表1　SZ310-U00仕様

型番	SZ310-U00
FPGA	Virtex-II Pro (XC2VP4-FG256)
フラッシュROM	8Mバイト
RAM	32Mバイト
ネットワーク	10Base-T/100Base-TX
拡張I/Oピン	70ピン
OS	uClinux

表2　今回使用した開発環境 (Windows版)

ソフトウェア	バージョン
ISE Project Navigator	8.2i
EDK Platform Studio	8.2i

表3　EDKプロジェクトに追加するファイル

top.vhd	FPGAのトップ・モジュール． 入出力ピン，クロックの設定などを記述する
top.ucf	FPGAのピン・アサイン，信号条件を指定する
xps_proj/xps_proj.xmp	EDKのプロジェクト・ファイル． ISEからEDKを起動する

● キャラクタ型ドライバの構造

●初期化

デバイス・ドライバの初期化は，init_module()がカーネルから呼び出されたところから始まります．init_module()ではregister_chardev()を呼び出して，デバイスのメジャー番号とマイナ番号，ファンクション・テーブルを登録します．**リスト1**にファンクション・テーブルを表すfile_operations構造体を示します．

●ファンクション

カーネルは，アプリケーションのシステム・コールに対応してデバイス・ドライバの処理関数を呼び出します．処理関数は複数あり，それらは「ファンクション・テーブル」というテーブルにまとめて登録されています．処理関数を呼び出すときには，ファンクション・テーブルを参照してから呼び出されます．

2. ターゲット・システム

● ボードの仕様と開発環境

今回使用するSUZAKU-Vは，PowerPC405を内蔵したFPGA (Xilinx社Virtex-II Pro) を搭載した小型ボードです．標準OSとしてLinuxを採用しており，開発環境も一通りそろっています．ユーザがFPGA内部に独自回路を追加し，70本の拡張I/Oピンを自由に使えます．

SUZAKUの仕様を**表1**に示します．

FPGAの開発には**表2**に示すXilinxの開発ツールを使います．筆者の環境は最新ではありませんが，最新版でもそれほど操作に変わりはないと思います．

ソフトウェアの開発にはLinuxを使用します．筆者は「RedHat Linux 9」を使いました．必要なツールはSUZAKU公式サイトからダウンロードします[注2]．

FPGAの開発ツールはWindows版なので，結果的にWindowsマシンとLinuxマシンと2台のパソコンが必要になります．

● FPGA開発環境の検証

FPGA開発用Windowsマシンには，あらかじめXilinx社製デバイスの開発ツールをインストールしておきます．さらにFPGAのひな型はXilinxの6.3i用の「suzaku-v-20061017.zip」を使います．このプロジェクト・ファイ

注2：SUZAKU　開発ツールのダウンロード先，http://suzaku.atmark-techno.com/downloads/all

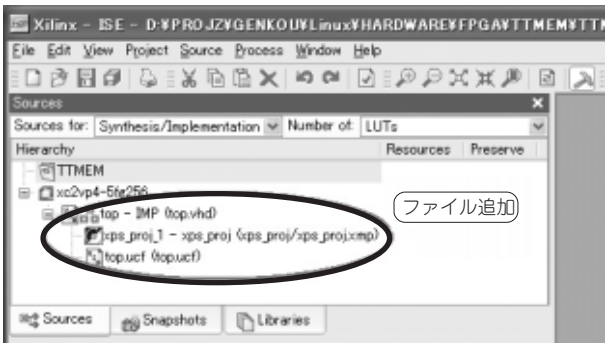

図1　EDKプロジェクトへ追加

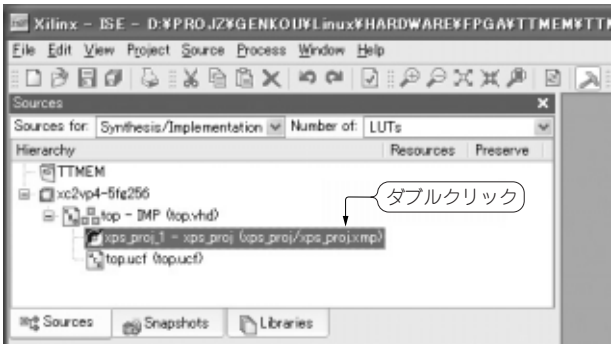

図2　EDKプロジェクトの起動