

第8章

μClinux 対応学習用ボードにブート・ローダを移植する

ブート・ローダ Das U-Boot の実装 (Blackfin 編)

川本 泰久

Das U-Boot (以下 U-Boot) は、RedBoot と同様に、複数のアーキテクチャに対応したフリーのブート・ローダで、ライセンスは GPL でリリースされています。ここでは、実際に U-Boot を動作させるには何をしたらよいのかを、CQ 出版社の「組み込みシステム開発評価キット (以下評価キット)」^{注1}やデバイスドライバーズの「E!Kit-BF533」といった学習用組み込みボードを使って説明します。「E!Kit-BF533」は、Analog Devices 社製の Blackfin プロセッサ「ADSP-BF533」を搭載した、uClinux 2.6 系カーネルを移植済みの評価用ボードです (写真1, 表1)。組み込みシステム開発評価キットと接続して使用できます。なお、Blackfin プ

ロセッサの詳細仕様やデータブックなどは、同社の Web サイト (<http://www.analog.com/jp/embedded-processing-dsp/blackfin/processors/index.html>) から入手可能です。

1. 開発環境の準備

● 開発環境の準備

ここでは、図1のような機器構成を例に説明します。E!Kit-BF533 は評価キットに実装してあり (写真1)、ホスト・パソコンとはシリアル・ケーブルと JTAG ケーブルで接続されています。

開発環境として、Linux 上で動作する GNU Tool chain を使用することにします。GNU Toolchain は、Blackfin Linux Project (<http://blackfin.uclinux.org/gf/>) から入手可能です。「GNU Tool

注1: 「組み込みシステム開発評価キット」は、FPGA や SDRAM、フラッシュ ROM、グラフィックス出力やオーディオ入出力などの A/V 機能、PCI、LAN、IDE、COM、USB、PS/2 などの I/O 機能を搭載した教育用ボードである。

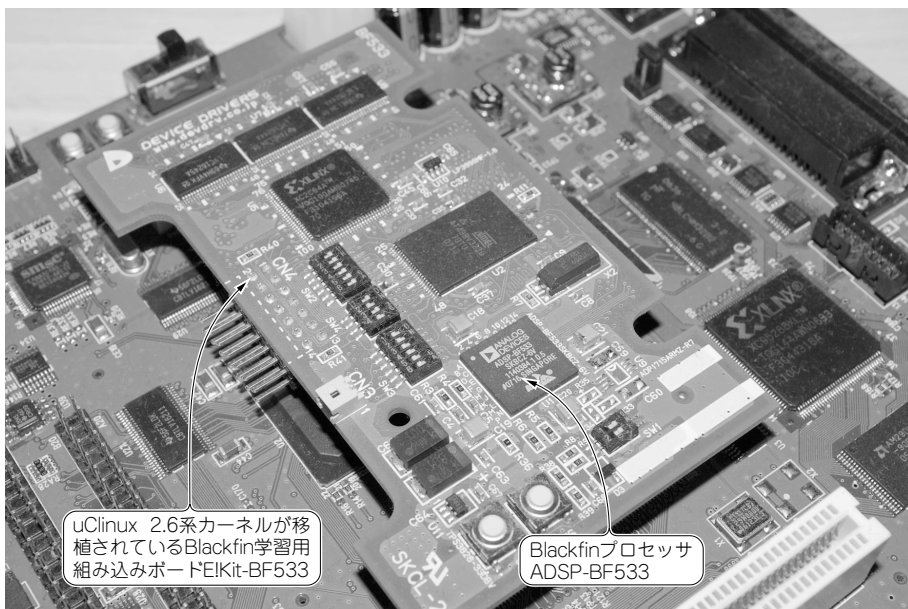


写真1

E!Kit-BF533 の外観

「組み込みシステム開発評価キット」の上に実装したところ。本ボードは、組み込みシステム開発評価キットのオプション CPU カード用コネクタを備えている。

表1
E!Kit-BF533の概略仕様

CPU	Analog Devices 社製 ADSP-BF533 600MHz 16K バイト 命令キャッシュ/ 32K バイト データ・キャッシュ
SDRAM	64M バイト, 16 ビット
フラッシュROM	4M バイト, 16 ビット
シリアル・インターフェース	UART × 1
CPLD	Xilinx XC2C64A 100 ピン
JTAG コネクタ	14 ピン
拡張ソケット	120 ピン × 2 (CQ 出版社製「組み込みシステム開発評価キット」 に対応, 独自拡張ボードを開発予定)
CPU 温度特性	動作時: 0℃ ~ 50℃ (周辺温度)
電源	+3.3V ± 5%, max 1.0A
大きさ	100mm × 60mm
対応 OS	uClinux 2.6系カーネル (2.6.18を移植, 製品に添付) TOPPERS/JSP (μITRON4.0仕様準拠, 動作確認済み) .NET Framework にも対応予定

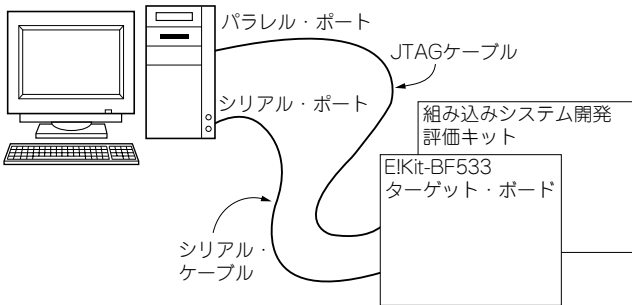


図1 DAS U-Boot の実装を試したシステム

chain」右横の「Releases」部分をクリックすれば該当ページにたどり着けます。なお、本章執筆時と同じバージョンをダウンロードしたい場合は、さらにパッケージ名「Blackfin Toolchain」をクリックしてください。各リリース一覧にたどり着けます。ダウンロード後、図2のように解凍し、パスを設定すればインストールが完了します。

● ソース・ファイルの入手

U-Bootのソース・ファイルは、DENX Software EngineeringのWebサイト (<http://www.denx.de/wiki/U-Boot/WebHome>) からオリジナルのものを入

```
$ cd ~
$ tar jxf u-boot-1.1.6.tar.bz2
```

図3 U-Boot ソース・ファイルの設定

手するか、Blackfin Linux Projectから入手します。本章ではBlackfin Linux Projectで公開されているものを使用しています。「Das U-boot」右横の「Releases」部分をクリックすれば該当ページにたどり着けます。なお、本章執筆時と同じバージョンをダウンロードしたい場合は、さらにパッケージ名「Das U-Boot」をクリックしてください。各リリース一覧にたどり着けます。ソース・ファイルを入手後、図3のように解凍すれば準備完了です。

開発環境の準備が整ったので、E!Kit-BF533にU-Bootを実装します。今回は、ハードウェア構成に近いbf533-stampのソースを基に実装することにしましょう。

```
$ su
Password
# cd /
# tar zxf blackfin-toolchain-07r1.1-3.i386.tar.gz
# exit

$ cd ~
$ vi .bash_profile

PATH=$PATH:$HOME/bin:/export/local/bin:/opt/uclinux/bfin-uclinux/bin/
```

図2 GNU Toolchain の設定

表2 ADSP-BF533のブート・モード

ブート・モード	解説
0	16ビット・バス幅の外部メモリ (0x20000000) から実行
1	オンチップ・ブートROMが実行され、8ビット/16ビット・バス幅の外部メモリからコードを読み込み実行
2	オンチップ・ブートROMが実行され、SPI経由でコードが書き込まれ実行
3	オンチップ・ブートROMが実行され、SPIシリアルEEPROMからコードを読み込み実行

2. CPUの動作と初期化

● リセット直後の動作を理解しよう

ADSP-BF533にはいくつかのブート・モードがあります(表2)。例えばモード0では、リセット直後にリセット・イベントが発生して、0x20000000から実行します。図4のメモリ・マップを見るとわかりますが、ここにはフラッシュROMが接続されており、U-Bootのコード領域が配置されています。

リンカ・スクリプト(リスト1)を見ると、リセット直後に実行されるリセット・イベント・コード(リスト2)をコード領域の先頭に配置するように定義しています。リンカ・スクリプトで定義されている先頭アドレスはメイン・メモリ上のアドレス(0x03FC0000)になっていますが、これは最終的にU-Bootはメイン・メモリ上で実行されるからです。BlackfinのJUMP/CALL命令はすべて相対アドレスなので、このままフラッシュROM上で実行可能です。

リセット・イベント・コード内では、まず各種DAG(データ・アドレス発生器)、DATA、ADDRESSレジスタの初期化、リセット要因の確認、イベント・ベクトル・テーブル、スタック・ポインタの初期化を行います。この段階ではまだメイン・メモリ(SDRAM)が動作していないので、スタック領域としてADSP-BF533内部のスクラッチ・パッドRAMを

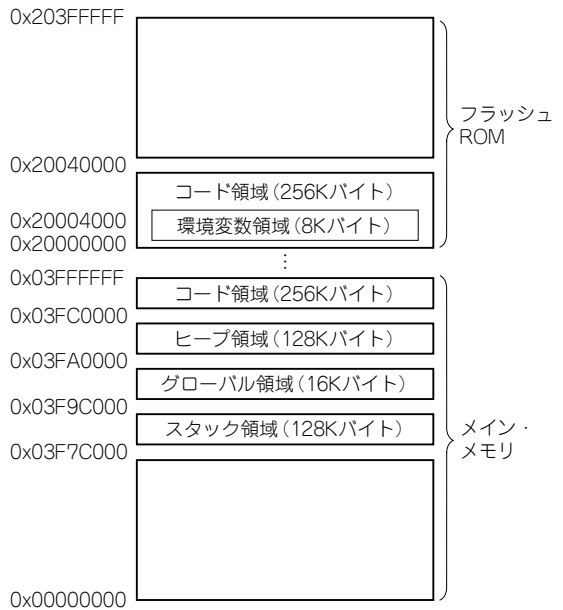


図4 U-Bootのメモリ・マップ

使用しています。

その後、メイン・メモリ(SDRAM)を動作させるために必要なクロックやメモリ・コントローラを初期設定して(リスト3)、フラッシュROM上のコード領域をメイン・メモリのコード領域にコピーし、スタック・ポインタを初期化してスーパーバイザ・モードを維持しつつ、リセット・イベントから抜けます。これ以降はメイン・メモリにコピーしたコードが実行されるようになります。U-Bootをメイン・メモリで動作させることで、U-Boot上でU-Boot自身をアップデートすることが可能になっています。

● CPUの初期化で行われていること

リセット・イベント・コードから抜けた後は、リスト4に示すように、初期化ルーチン(リスト5)が呼び出されます。CPUが動作するのに必要な初期化はすでにリセット・イベント・コードで行われているので、ここではU-Bootのアーキテクチャに依存する部分の処理が行われます。

まずキャッシュ・テーブルとグローバル領域に格納

リスト1

リンカ・スクリプト(u-boot-1.1.6/board/bf533-stamp/u-boot.ldsの一部)

```
SECTIONS
{
    . = CFG_MONITOR_BASE;
    .text :
    {
        cpu/bf533/start.o (.text)
        cpu/bf533/start1.o (.text)
        cpu/bf533/traps.o (.text)
        (中略)
    }
}
```

プリプロセスで適切な値に変換される例: 0x3FC0000
 =((64×1024×1024)-0x40000)

リセット後に実行されるコードを配置(リスト2参照)

リスト2 リセット・イベント・コード (u-boot-1.1.6/cpu/bf533/start.Sの一部)

```

.text
_start:
start:
_stext:
    R0 = 0x32;
    SYSCFG = R0;
    SSYNC;

    r1 = 0; /* Data registers zero'd */
    r2 = 0;
    (中略)

    lc0 = r0;
    lc1 = r0;
    SSYNC;

    /* Check soft reset status */
    p0.h = SWRST >> 16;
    p0.l = SWRST & 0xFFFF;
    r0.l = w[p0];

    cc = bittst(r0, 15);
    if !cc jump no_soft_reset;

    /* Clear Soft reset */
    r0 = 0x0000;
    w[p0] = r0;
    ssync;

no_soft_reset:
    nop;

    p0.h = (EVT_EMULATION_ADDR >> 16);
    p0.l = (EVT_EMULATION_ADDR & 0xFFFF);
    p0 += 8;
    p1 = 14;
    r1 = 0;
    LSETUP(4,4) lc0 = p1;
    [ p0 ++ ] = r1;

    p0.h = hi(SIC_IWR);
    p0.l = lo(SIC_IWR);
    r0.l = 0x1;
    w[p0] = r0.l;
    SSYNC;

    sp.l = (0xffb01000 & 0xFFFF);
    sp.h = (0xffb01000 >> 16);

    call init_sdram;
    call get_pc;

offset:
    r2.l = offset;
    r2.h = offset;
    r3.l = start;
    r3.h = start;

    r1 = r2 - r3;
    r0 = r0 - r1;
    p1 = r0;

    p2.l = (CFG_MONITOR_BASE & 0xffff);
    p2.h = (CFG_MONITOR_BASE >> 16);

    p3 = 0x04;
    p4.l = ((CFG_MONITOR_BASE + CFG_MONITOR_LEN)
            & 0xffff);
    p4.h = ((CFG_MONITOR_BASE + CFG_MONITOR_LEN)
            >> 16);

loop1:
    r1 = [p1 ++ p3];
    [p2 ++ p3] = r1;
    cc=p2=p4;
    if !cc jump loop1;

    r0.h = (CONFIG_STACKBASE >> 16);
    r0.l = (CONFIG_STACKBASE & 0xFFFF);
    sp = r0;
    fp = sp;

    p0.l = (EVT_IVG15_ADDR & 0xFFFF);
    p0.h = (EVT_IVG15_ADDR >> 16);

    p1.l = _real_start;
    p1.h = _real_start;
    [p0] = p1;

    p0.l = (IMASK & 0xFFFF);
    p0.h = (IMASK >> 16);
    r0.l = LO(IVG15_POS);
    r0.h = HI(IVG15_POS);
    [p0] = r0;
    raise 15;
    p0.l = WAIT_HERE;
    p0.h = WAIT_HERE;
    reti = p0;
    rti;

WAIT_HERE:
    jump WAIT_HERE;

.global _real_start;
_real_start:
    [ -- sp ] = reti;
    (中略)
    p0.l = _start1;
    p0.h = _start1;
    jump (p0);
    (中略)

get_pc:
    r0 = rets;
    rts;
    
```

システム設定レジスタ (SYSCFG) 設定

リセット後、各種 DAG, DATA, ADDRESSレジスタの値が不定なので0に初期化

ソフトウェア・リセットかどうかの判定

ソフトウェア・リセットの場合は要因クリア

イベント・ベクトル・テーブル初期化

スタック・ポインタをスクラッチ・パッドRAMに設定

メモリ・コントローラ初期化ルーチン呼び出し(リスト3参照)

フラッシュROM上のstart:レベル(U-Boot領域の先頭)のアドレスをp1レジスタに格納

フラッシュROM上のU-Boot領域をメイン・メモリ上にコピー

スタック・ポインタ初期化

スーパーバイザ・モードを維持しつつリセット・イベントから抜け real_start:から実行する(これ以降はメイン・メモリ上で実行される)

_start:へジャンプ(リスト4参照)

される構造体(ここにはさまざまな情報が格納される)の初期化が行われ、リセット・イベント・コードでは初期化されていない割り込みコントローラやUARTコントローラなどの初期化を行います。その後、キャッシュが有効になっている場合は設定を行い、起動メッセージの表示やヒープ領域の初期化などが行われます。そして最後にアーキテクチャに依存しないコードが実行され、U-Bootが起動します。

ここまでの流れを見るとわかるように、U-Bootを新しいCPUに実装する場合は、リセット・イベント・

コードと初期化コードを変更する必要があります。一方、すでに実装されているCPUの場合は、リセット・イベント・コードの変更だけで済みそうです。

3. U-Bootのコンパイルと動作確認

● いよいよコンパイルする

E!Kit-BF533へU-Bootを実装するために変更した箇所をリスト6にパッチ形式で示します。変更箇所を見るとわかるように、クロック周波数やメモリ容量、種