(機械で日本語に翻訳)

Getting started with Xillinux for Zynq-7000 v2.0

Xillybus Ltd.

www.xillybus.com

Version 4.0

この文書はコンピューターによって英語から自動的に翻訳されているため、言語が不明瞭になる可能性があります。このドキュメントは、元のドキュメントに比べて少し古くなっている可能性もあります。

可能であれば、英語のドキュメントを参照してください。

This document has been automatically translated from English by a computer, which may result in unclear language. This document may also be slightly outdated in relation to the original.

If possible, please refer to the document in English.

1

序章

1.1 Xillinux ディストリビューション

Xillinux は、 Zynq-7000 デバイス用の完全なグラフィカルな Lubuntu 16.04ベースの Linux ディストリビューションであり、混合ソフトウェア/ logic プロジェクトの迅速な開発のためのプラットフォームとして意図されています。現在サポートされているボードは、 Z-Turn Lite、 Zedboard、 MicroZed 、および Zyboです。

他の Linux ディストリビューションと同様に、 Xillinux は、 Linuxを実行しているパーソナルデスクトップコンピューターとほぼ同じ機能をサポートするソフトウェアのコレクションです。一般的な Linux ディストリビューションとは異なり、 Xillinux には、ハードウェア logicの一部、特に VGA アダプターも含まれています。

Z-Turn Lite、 Zedboard 、および Zyboを使用すると、ディストリビューションは、従来のキーボード、マウス、およびモニターの設定用に編成されます。また、 USB UART ポートからのコマンドライン制御も可能ですが、この機能は主に問題を解決するために使用できます。

VGA/DVI 出力がない MicroZedで使用する場合、 USB UART のみが consoleとして使用されます。

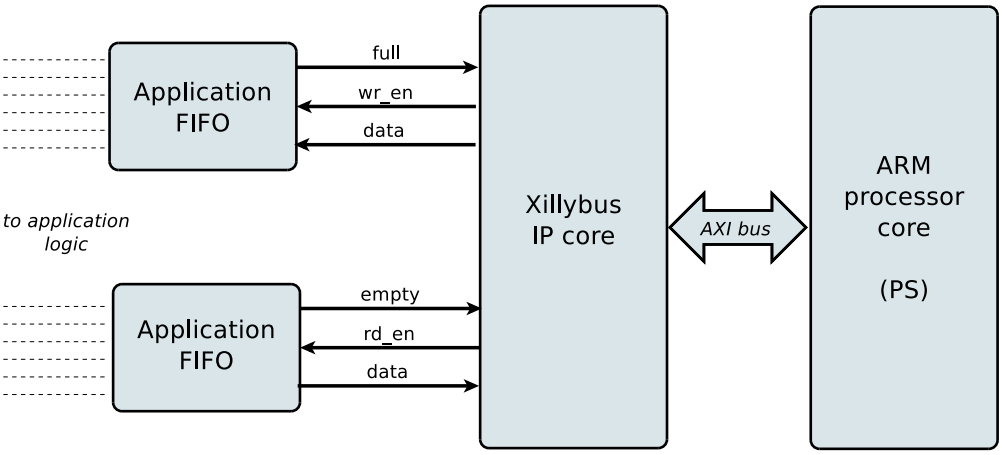
Xillinux は、デバイスの FPGA logic fabric と ARM processorで実行されているプレーン user space applications を統合するためのキックスタート開発プラットフォームでもあります。 Xillybus IP core と driverが含まれているため、 FPGA logic と Linuxベースのソフトウェアが連携するアプリケーションの design を完成させるために必要なのは、プログラミングと logic design の基本的なスキルだけです。

バンドルされた Xillybus IP cores は、アプリケーション設計者にシンプルでありながら効率的な作業環境を提供することにより、 kernel programming の低レベルの内部および application logic と processor間のインターフェイスを処理する必要性を排

除します。

1.2 Xillybus IP core

Xillybus は、 FPGA と Linux または Microsoft Windowsを実行する host 間のデータ転送用の DMAベースのエンドツーエンド ソリューションです。 FPGA logic の設計者だけでなく、ソフトウェアのプログラマーにもシンプルで直感的なインターフェイスを提供します。 AMBA bus (AXI3/AXI4) とインターフェイスする ARMベースの processorsだけでなく、 PCI Express bus を基礎となるトランスポートとして使用するパーソナル コンピューターおよび embedded システムでも使用できます。



上に示したように、 FPGA 上の application logic は、標準の FIFOsとのみ対話する必要があります。

たとえば、図の下部の FIFO にデータを書き込むと、 Xillybus IP core は、 FIFOのもう一方の端でデータを送信できることを認識します。間もなく、 IP core は FIFOからデータを読み取り、それを hostに送信して、 userspace softwareで読み取り可能にします。データ転送メカニズムは、 FIFOと相互作用するだけの FPGAの application logic に対して透過的です。

一方、 Xillybus IP core は、 AXI busを利用してデータフローを実装し、 processor coreの busで DMA 要求を生成します。

host 上のアプリケーションは、 named pipesのように動作する device files と対話します。 Xillybus IP core と driver は、 FPGAs の FIFOs と hostの関連する device files の間でデータを効率的かつ直感的に転送します。

IP core は、オンライン Web アプリケーションを使用して、顧客の仕様に従って即座に構築されます。 streamsの数、方向、およびその他の属性は、 designの帯域幅

パフォーマンス、同期、およびシンプルさの間で最適なバランスを実現するために、お客様が定義します。

このガイドの説明に従って準備を行った後、 http://xillybus.com/custom-ip-factoryでカスタム IP core をビルドしてダウンロードすることをお勧めします。

このガイドでは、 Xillybus IP core を含む Xillinux ディストリビューションを迅速にセットアップする方法について説明します。この IP core は、実際のアプリケーション シナリオのテストのために、ユーザー提供のデータ ソースとデータ コンシューマに接続できます。これはデモンストレーション キットではありませんが、便利なタスクをそのまま実行できるフル機能の starter designです。

既存の IP core を特別なアプリケーション用に調整されたものに置き換えるのは簡単なプロセスであり、1 つのバイナリ ファイルを置き換え、1 つのモジュールをインスタンス化する必要があります。

Xillybus IP core の使用に関する詳細は、次のドキュメントに記載されています。

- Getting started with Xillybus on a Linux host
- Xillybus host application programming guide for Linux
- The guide to defining a custom Xillybus IP core

興味のある方のために、 Xillybus IP core の実装方法に関する簡単な説明が Xillybus host application programming guide for Linuxの付録Aにあります。

2

前提条件

2.1 ハードウェア

Zynq用の Xillybus による Linux ディストリビューションであるXillinuxは、現在次のボードをサポートしています。

- Z-Turn Lite （および使用可能な Zynq デバイス）
- Zedboard
- 7010 MicroZed。 7020 MicroZed は、マイナーな修正でサポートされています。セクション 5.5を参照してください。
- Zybo

Z-Turn Lite ボードを購入しようとしている場合は、このボードのこの web page にアクセスして、アイテムの選択を支援することをお勧めします。

上記にリストされていないボードの所有者は、独自のハードウェアでディストリビューションを実行できますが、特定の変更が必要になる場合があります。詳細については、セクション 5.2をご覧ください。

ボード（MicroZed を除く）をモニター、キーボード、マウスを備えたデスクトップコンピューターとして使用するには、次のアイテムが必要です。

- ボードの出力に応じて、アナログ VGA または HDMI 入力を備えた VESA準拠の 1024x768 @ 60Hz を表示できるモニター（つまり、事実上すべてのPCモニター）。
- モニター用の VGA または HDMI ケーブル（該当する場合）
- USB キーボード

- USB マウス
- USB hub、キーボードとマウスが1つの USB プラグに結合されていない場合

Z-Turn Lite ボードにはモニターへの出力がないことに注意してください。したがって、デスクトップの使用シナリオでは、 HDMI ポートを持つ Z-Turn Lite IO Cape boardを接続する必要があります。

Zybo を使用する場合、モニターは VGA ポートだけでなく HDMI ポートにも接続できます。 Zedboardの HDMI 出力ポートはサポートされていません。

USB hubが不要になり、 USB ケーブルを誤って引っ張った結果としてボード上の USB ポートが物理的に損傷するのを防ぐため、ワイヤレスキーボード/マウスの組み合わせをお勧めします。

Zedboard および Z-Turn Liteでは、キーボードとマウスの接続は、Micro Bからタイプ Aのメス USB ケーブルを介して行われます。このケーブルは、 Zedboard および場合によっては Z-Turn Lite に付属しています（購入したアイテムによって異なります）。

他の2つのボードでは、標準の USB タイプAメスコネクタ（ PCの USB プラグなど）を周辺機器の接続に使用できます。

また必要：

- 4GB 以上の信頼性の高い SD カード（ Zedboardの場合）または MicroSD （ Z-Turn Lite、 MicroZed 、および Zyboの場合）、最も好ましくは Sandisk製のカード。 Xillinuxで使用すると問題が報告されているため、（おそらく）ボードに付属のカードはお勧めしません。
- 推奨： image および boot file をカードに書き込むための、 (Micro)SD カードと PCの間の USB アダプター。 PCコンピュータに SD カード用のスロットが組み込まれている場合、これは不要な場合があります。 Zynq ボード自体を使用して SD カードに書き込むこともできますが、これはやや困難です。

2.2 ディストリビューションのダウンロード

Xillinux ディストリビューションは、 Xillybus サイトのダウンロードページからダウンロードできます。

http://xillybus.com/xillinux/

ディストリビューションは2つの部分で構成され、2つの別々のファイルとしてダウンロードされます。

- 起動時に Linux に表示される file system で構成される (Micro)SD カードの raw image
- boot partitionにデータを入力するために、 Xilinxのツールで implementation を実行するためのファイルのセットである boot partition kit。

詳細については、セクション 3を参照してください。

ディストリビューションには、 processor と logic fabric間の簡単な通信のための Xillybus IP core のデモが含まれています。この demo bundle の特定の構成は、単純なテストを目的としているため、特定のアプリケーションでは比較的パフォーマンスが低下する可能性があります。

カスタム IP cores は、 IP Core Factory Webアプリケーションを使用して構成、自動構築、およびダウンロードできます。このツールの使用については、http://xillybus.com/custom-ip-factory にアクセスしてください。

Xillybus IP coreおよび Xillinux ディストリビューションを含む、ダウンロードされたバンドルは、この使用法が “evaluation”という用語と合理的に一致する限り、無料で使用できます。これには、エンドユーザー designsへの IP core の組み込み、実際のデータの実行、およびフィールドテストが含まれます。 IP core の使用目的が特定のアプリケーションに対する機能と適合性を評価することである限り、 IP core の使用方法に制限はありません。

2.3 開発ソフトウェア

Vivado 2014.4 **以降**は、 Xillinux ディストリビューションの logic design の compilation に使用できます。

7z007s または 7z014s デバイスを使用する場合は、これらをサポートする Vivado リビジョンが必要です（ Vivado 2016.4 以降など）。

ソフトウェアは、 XilinxのWebサイト（http://www.xilinx.com）から直接ダウンロードできます。

Vivadoのエディションのいずれかが適しています。

- WebPack Editionは、目的のデバイスがカバーされていることを前提として、ライセンス料なしで無制限にダウンロードして使用できます。 Z-Turn Lite、Zedboard、 MicroZed 、および Zybo に対応するすべてのデバイスがこのエディションの対象となります。
- Design Edition。購入したライセンスが必要です（ただし、30日間の試用版が利用可能です）。

- 購入したボードで特別にライセンスされている可能性があるため、特定のZynq デバイスに限定されているエディション。
- System および WebPack Editionの機能のスーパーセットを提供する他のエディションも同様に問題ありません。

これらのエディションはすべて、 Zynq用の Xillybus を実装するために必要なXilinx提供の IP cores をカバーしており、追加のライセンスは必要ありません。

2.4 FPGA designの使用経験

Z-Turn Lite、 Zedboard、 MicroZed 、または Zyboを使用する場合、プラットフォームでディストリビューションを実行するために、 FPGA design の経験は必要ありません。別のボードを使用するには、 Xilinxのツールの使用に関する知識と、 Linux kernelに関連する基本的なスキルが必要です。

ディストリビューションを最大限に活用するには、 logic design の手法を十分に理解し、 HDL 言語（Verilog または VHDL）を習得する必要があります。それでも、Xillybus ディストリビューションは、実験するための簡単なスターター design を提供するため、これらを学習するための良い出発点です。

3

Xillinuxの構築

3.1 概要

Xillinux ディストリビューションは、単なるデモではなく、開発プラットフォームとして意図されています。カスタム logic の開発と統合のためのすぐに使用できる環境は、ハードウェアで実行するための準備中に構築されます。したがって、最初のテスト実行の準備時間は少し長くなります（通常、30分で、そのほとんどはXilinxのツールを待つことで構成されます）。ただし、この長い準備により、カスタム logicを統合するサイクルが短縮されます。

(Micro)SD カードからの Xillinux ディストリビューションの boot プロセスを成功させるには、次の2つのコンポーネントが必要です。

- boot loaders、 FPGA パーツ用の configuration bitstream （ PLとして知られている）、および Linux kernelの boot 用のバイナリファイルで構成される boot partition内の FAT32 filesystem 。
- Linuxによってマウントされた ext4 root file system 。

Xillinux のダウンロードされた raw image には、ほとんどすべてがすでにセットアップされています。 boot partitionには3つのファイルがあり、そのうちの1つはXilinxのツールで生成する必要があり、2つは boot partition kitからコピーされたものです。

このセクションでは、 (Micro)SD を準備するためのさまざまな操作について段階的に詳しく説明します。

この手順は、以下の手順で構成されています。これらの手順は、以下に概説する順序で実行する必要があります。

- boot partition kit を解凍する

- メインの PL （FPGA）プロジェクトの implementation を実行する
- Xillinux image を (Micro)SD カードに書き込む
- (Micro)SD カードの boot partition に3つのファイルをコピーする

他のボードを操作する方法については、 5.2項で説明しています。

3.2 boot partition kitを解凍する

以前にダウンロードした xillinux-eval-*board-XXX*.zip ファイルを作業ディレクトリに解凍します。

重要:

作業ディレクトリへのパスに空白を含めることはできません。特に、 *Windows Desktop* のパスには *"Documents and Settings"*が含まれているため、 *Windows Desktop* は不適切です。

バンドルは、次のディレクトリ（またはそれらの一部）で構成されています。

- verilog –メイン logic のプロジェクトファイルと Verilog （ 'src' サブディレクトリ内）のいくつかのソースが含まれています
- vhdl –メイン logic のプロジェクトファイルといくつかのソースファイルが含まれています。編集する VHDL のファイルは 'src' サブディレクトリにあります
- blockdesign –このディレクトリには、 Block Design Flow に関連するファイルが含まれています（ The guide to Xillybus Block Design Flow for non-HDL usersを参照）。
- cores – Xillybus IP coresのプリコンパイルされたバイナリ
- system – processor関連の logicを生成するためのディレクトリ
- runonce –汎用 logicを生成するためのディレクトリ。一部のバンドルには存在しない場合があります。
- bootfiles – boot partitionにコピーされる2つのボード固有のファイルが含まれています。
- vivado-essentials – Vivadoで使用するための processor関連および汎用 logic の定義ファイルとビルドディレクトリ。

Z-Turn Lite、 Zedboard、 MicroZed 、および Zybo ボードで使用可能なバンドルがあります。別のボードを使用する場合は、セクション 5.2に記載されている問題に加えて、 constraints ファイル vivado-essentials/xillydemo.xdcを適宜編集する必要があります。

vhdl ディレクトリには Verilog ファイルが含まれていますが、ユーザーが編集する必要はありません。

Xillybus IP core とのインターフェースは、それぞれの 'src' サブディレクトリ内の xillydemo.v または xillydemo.vhd ファイルで行われます。これは、独自のデータソースとデータ コンシューマーで Xillybus を試すために編集するファイルです。

3.3 bitstream ファイルの生成

3.3.1 序章

Vivado は、比較的複雑な構造で多くの中間ファイルを生成するため、プロジェクトを管理することが困難になります。バンドル内のファイル構造をコンパクトに保つために、 Vivado プロジェクトを作成するために Tcl の script が提供されています。この script は、新しいサブディレクトリ "vivado"を作成し、必要に応じてこのディレクトリにファイルを追加します。

プロジェクトは、 src/ サブディレクトリ内のファイルに依存しています（これらのファイルのコピーは作成されません）。 processor、その相互接続と周辺機器、および logic で使用される FIFOs は vivado-essentials/で定義されており、プロジェクトの implementation中に Vivado によって中間ファイルも入力されます。

プロジェクトの implementation は、 Verilog、 VHDL、または Block Design Flowに基づくことができます（後者は The guide to Xillybus Block Design Flow for non-HDL usersで説明されています）。

3.3.2 目的の Zynq パーツの選択（Z-Turn Lite のみ）

この手順は、 Z-Turn Lite用の demo bundle で**のみ**必要です。

テキストエディタを使用して、バンドルの root directory で select_part.tcl を開きます。このファイルの最後の4行は、 Vivado プロジェクトが作成される Zynq パーツを設定するための Tcl コマンドです。これらの4行は、 "#" 文字でコメント化されています。

Z-Turn Lite ボード上の Zynq パーツを選択するには、これらの行の1つから1つの "#" 文字のコメントを解除します。

後で別の Zynq パーツを使用する場合は、それに応じて Vivado プロジェクトの設定を変更し、プロジェクトを再実装します。 select_part.tcl はプロジェクトの生成中にのみ参照されるため、後で変更しても効果はありません。

3.3.3 xillydemo.vhd の準備（VHDL プロジェクトのみ）

この手順は、 VHDL が選択されている場合にのみ必要であり、 demo bundle は Z-Turn Lite**以外**のすべてのボードを対象としています。

vhdl/src/xillydemo.vhd を編集して、 Xillydemoのエンティティポートリストの先頭にある次の3行を**削除する**必要があります。

```
    PS_CLK : IN std_logic;
    PS_PORB : IN std_logic;
    PS_SRSTB : IN std_logic;
```

も、アーキテクチャ定義の次の行の**コメント**を解除します（ “--” コメントマークを削除します）。

```
  -- signal PS_CLK :  std_logic;
  -- signal PS_PORB :  std_logic;
  -- signal PS_SRSTB :  std_logic;
```

Verilog ソースファイルを変更する必要はありません。

3.3.4 Vivado プロジェクトの生成

Vivado 2014.4 以降を起動します。

プロジェクトを開いていない状態で、 Tools > Run Tcl Script... を選択し、好みに応じて verilog/、 vhdl/ 、または blockdesign/ サブディレクトリで **xillydemo-vivado.tcl** を選択します。一連のイベントは1分未満で発生します。プロジェクトの展開が成功したかどうかは、 Vivadoのウィンドウの下部にある[“Tcl Console”]タブを選択し、それが次のように表示されていることを確認することで確認できます。

```
INFO: Project created: xillydemo
```

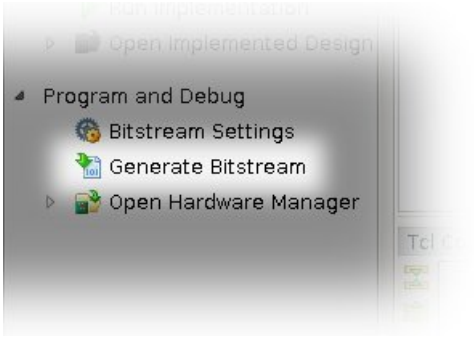
これが Tcl consoleの出力の最後の行でない場合は、問題が発生しています。

この段階では Critical Warnings が発生する可能性がありますが、エラーは発生しません。ただし、プロジェクトがすでに生成されている場合（つまり、 script がすでに実行されている場合）、 script を再度実行しようとすると、次のエラーが発生します。

```
ERROR: [Common 17-53] User Exception: Project already exists on disk,
       please use '-force' option to overwrite:
```

3.3.5 プロジェクトのImplementation

プロジェクトが作成されたら、 implementationを実行します。左側の Flow Navigator bar で[“Generate Bitstream”]をクリックします。



synthesis と implementationを起動してもよいかどうかを尋ねるポップアップウィンドウが表示される可能性があります。 “Yes”を選択してください。

Vivado は一連のプロセスを実行します。これには通常、数分かかります。いくつかの warnings が発行されており、そのうちのいくつかはクリティカルに分類される場合があります。エラーはないはずです。

bitstream が正常に生成されたことを通知するポップアップウィンドウが表示され、次に何をするかを選択できます。 “Cancel”の選択を含め、どのオプションでも問題ありません。

bitstream ファイル xillydemo.bitは、 vivado/xillydemo.runs/impl_1/にあります。

implementation が故障することは決してありません。ただし、言及する価値のあるエラー条件がいくつかあります。

- プレーサーは、 VHDL design上の “IO placement is infeasible” で失敗します。VHDLを搭載した implementation でこれが発生する場合は、 xillydemo.vhd が上記の必要に応じて編集されていることを確認してください。
- write_bitstream は、 PS_CLK、 PS_PORB 、および PS_SRSTB が指定されておらず、ルーティングされておらず、制約がないことを示す DRC エラーで失敗します。その後、 xillydemo.vhd が上記の必要に応じて編集されていることを確認してください。

- “Timing constraints weren’t met”というエラー。これは、カスタム logic が統合されている場合に発生する可能性があり、ツールが timing 要件を満たせなくなる可能性があります。これは、 design が構文的に正しいことを意味しますが、特定のパスを特定の clock レートおよび/または I/O 要件に関して十分に高速にするための修正が必要です。 design をより良い timing に修正するプロセスは、 *timing closure*と呼ばれることがよくあります。

  timing constraint の障害は一般に critical warningとしてアナウンスされ、ユーザーが FPGAの動作が保証されていない bitstream ファイルを作成できるようにします。このような bitstreamの生成を防ぐために、 timing の障害は、 routeの実行の最後に自動的に実行される小さな Tcl script、 “showstopper.tcl”によって、エラーのレベルに昇格されます。この安全対策をオフにするには、 Flow Navigatorの “Project Manager” の下にある “Project Settings” をクリックします。 “Implementation” ボタンを選択し、 “route_design”の設定まで下にスクロールします。次に、 tcl.postから showstopper.tcl を削除します。

- その他のエラーは、ユーザーが行った変更の結果である可能性が高く、ケースごとに処理する必要があります。

3.4 (Micro)SD と imageのロード

3.4.1 全般的

このタスクの目的は、 image file を (micro)SD カードに書き込むことです。このファイルの名前は xillinux-2.0.img.gzで、圧縮ファイル (gzip 形式) としてダウンロードされます。

(Micro)SD カードのこの image は、 bootの準備ができています。ただし、カードへの書き込み後に追加される 3 つのファイルを除きます。

この image は圧縮解除してから、 (Micro)SD カードの最初の sector 以降に書き込む必要があります。これを達成するためのいくつかの方法とツールがあります。次に、いくつかの方法を提案します。

image には、 partition table、初期 boot filesを配置するための部分的に実装された FAT file system 、および ext4 タイプの Linux root file system が含まれています。2番目の partition は、ほとんどすべての Windows コンピューターで無視されるため、 (Micro)SD カードの容量は非常に小さいように見える場合があります（16 MB 程度）。

full disk image の作成は、普通のコンピューターユーザーを対象とした操作ではないため、 Windows コンピューターでは特別なソフトウェアを使用し、 Linuxでは特

別な注意を払う必要があります。次の段落では、いずれかのオペレーティングシステムでこれを行う方法について説明します。

(Micro)SD カード（またはコンピューターの専用スロット）用の USB アダプターがない場合は、 3.4.4項で説明されているように、ボード自体を使用して imageを書き込むことができます。

重要:

image を (Micro)SD に書き込むと、含まれている可能性のある以前のコンテンツが完全に削除されます。 *image*の作成に使用したのと同じツールを使用して、既存のコンテンツのコピーを作成することを強くお勧めします。

3.4.2 image のロード（Windows）

Windowsでは、 USB Image Toolなどの imageをコピーするための特別なアプリケーションが必要です。このツールは、 USB アダプタを使用して (Micro)SD カードにアクセスする場合に適しています。

一部のコンピューター（特にラップトップ）には (Micro)SD スロットが組み込まれており、 Win32 Disk Imagerなどの別のツールを使用する必要がある場合があります。これは、 Windows 7を実行している場合にも当てはまります。

どちらのツールも、Web上のさまざまなサイトから無料でダウンロードできます。次のウォークスルーは、 USB Image Toolの使用を前提としています。

グラフィカルインターフェイスの場合は、 “USB Image Tool.exe”を実行します。メインウィンドウが表示されたら、 USB アダプターを接続し、左上に表示されるデバイスアイコンを選択します。左上の drop down menuで（ “Volume Mode”ではなく） “Device Mode” を使用していることを確認してください。 Restore をクリックし、ファイルタイプを “Compressed (gzip) image files”に設定します。ダウンロードした image file （xillinux.img.gz）を選択します。全体のプロセスは約4-5分かかるはずです。終了したら、デバイスをアンマウントし（「ハードウェアを安全に取り外す」）、プラグを抜きます。

一部のマシンでは、 GUI の実行に失敗し、ソフトウェアの初期化に失敗したというエラーが表示されます。その場合、代替コマンドラインを使用するか、 Microsoft .NET framework コンポーネントをインストールする必要があります。

または、コマンドラインから実行することもできます（ GUI の実行に失敗した場合の簡単な代替手段です）。これは2段階で行われます。まず、デバイスの番号を取得します。 DOS Windowでは、ディレクトリをアプリケーションが解凍された場所に変更して移動します（通常のセッションが続きます）。

```
C:\usbimage>usbitcmd l

USB Image Tool 1.57
COPYRIGHT 2006-2010 Alexander Beug
http://www.alexpage.de

  Device    | Friendly Name             | Volume Name | Volume Path | Size
 ------------------------------------------------------------------------------
      2448 | USB Mass  Storage Device  |             | E:\         |   4024 MB
```

（ "usbitcmd" の後の文字は文字 “l” であり、数字 “1”ではないことに注意してください）

これで、デバイス番号がわかったら、実際に書き込みを行うことができます（”restore”）。

```
C:\usbimage>usbitcmd r 2448 \path\to\xillinux.img.gz /d /g

USB Image Tool 1.57
COPYRIGHT 2006-2010 Alexander Beug
http://www.alexpage.de

Restoring backup to "USB Mass  Storage Device USB Device" (E:\)...ok
```

繰り返しますが、これには約4～5分かかります。そしてもちろん、番号2448を最初の段階で取得したデバイス番号に変更し、 `\path\to` を (Micro)SD カードの image がコンピューターに保存されているパスに置き換えます。

3.4.3 image のロード（Linux）

重要:

デバイスへの生のコピーは危険な作業です。些細な人為的エラー（通常は間違った宛先ディスクの選択）により、コンピューターのハードディスク内のすべてのデータが回復不能に失われる可能性があります。 *Enter*を押す前に考え、 *Linux*に慣れていない場合は、 *Windows* でこれを行うことを検討してください。

今述べたように、 (Micro)SD カードとして正しいデバイスを検出することが重要です。これは、 USB コネクタを接続し、メインログファイル（/var/log/messages または /var/log/syslog）で次のようなものを探すことによって行うのが最適です。

```
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] 7813120 512-byte logical blocks
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Write Protect is off
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel:  sdc: sdc1
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Attached SCSI removable disk
Sep  5 10:31:00 kernel: sd 1:0:0:0: Attached scsi generic sg0 type 0
```

出力はわずかに異なる場合がありますが、ここでのポイントは、 kernel が新しいディスクに付けた名前を確認することです。 上記の例の “sdc” 。

image fileを解凍します。

```
# gunzip xillinux.img.gz
```

image を (Micro)SD カードにコピーするのは簡単です。

```
# dd if=xillinux.img of=/dev/sdc bs=512
```

もちろん、フラッシュドライブであることがわかったディスクを指す必要があります。

重要:

/dev/sdc が例として示されています。コンピュータで認識されているデバイスと一致しない限り、このデバイスを使用しないでください。

そして確認する

```
# cmp xillinux.img /dev/sdc
cmp: EOF on xillinux.img
```

応答に注意してください。 image file で EOF に到達したという事実は、他のすべてが正しく比較され、フラッシュに実際に使用されているよりも多くのスペースがあることを意味します。 cmpが何も言わない場合（通常は良いと見なされます）、実際には何かが間違っていることを意味します。ほとんどの場合、デバイスに書き込むのではなく、通常のファイル “/dev/sdc” が生成されました。

3.4.4 imageをロードするための Zynq ボードの使用

上記の 3.4.3 の段落では、 image に Linux マシンと USB アダプタをロードする方法について説明しました。 Zynq ボード自体を使用して、ボードに付属のサンプル

Linux システムを実行できます。基本的に、 /dev/mmcblk0 を宛先デバイスとして (/dev/sdcの代わりに) 使用して、同じ手順に従うことができます。

これは、 boot プロセスが QSPI flashから実行される場合、および SD カード上のサンプル Linux システム（ボードに付属している場合）で正常に機能します。これは、 Xillinuxとは異なり、 RAMから完全に実行され、 boot の終了後に SD カードを使用しないためです。そのため、 bootに SD カードを使用した場合は、カードを引き出して、 image を書き込むために別のカードを挿入することができます。

Zynq ボードに (Micro)SD の image および boot partition ファイルへのアクセスを許可する方法は、好みと Linux の知識の問題です。ネットワーク経由でこれを行うにはいくつかの方法がありますが、最も簡単な方法は、これらのファイルをディスクオンキーに書き込み、 USB OTG ポートに接続することです。 disk-on-key を

```
> mkdir /mnt/usb
> mount /dev/sda1 /mnt/usb
```

ディスクオンキー上のファイルは、 /mnt/usb/で読み取ることができます。

Zedboardでは、 JP2 ジャンパーが取り付けられていることを確認して、 USB ポートに 5V 電源が供給されるようにします。

3.5 boot partitionへのファイルのコピー

この最終段階では、 bootに必要なファイルが配置されます。

- boot.bin および devicetree.dtb を boot partition kitの bootfiles/ サブディレクトリから (Micro)SD カードの boot partition （最初の partition）にコピーします。
- セクション 3.3 で生成された xillydemo.bit をコピーします（ verilog/ または vhdl/ サブディレクトリのどちらか選択した方から）。

これらのファイルをコピーする前に： (Micro)SDの image がカードに書き込まれたばかりの場合は、 USB アダプターを取り外してから、コンピューターに接続し直します。 Zynq ボードを使用して raw imageを書き込んだ場合は、 (Micro)SD カードをスロットから引き出して、再度挿入します。

これは、コンピューターが (Micro)SD カードの partition tableで最新であることを確認するために必要です。

Linux システムでは、最初の partition （ /dev/sdb1など）を手動でマウントする必要がある場合があります。ほとんどのコンピューターはこれを自動的に行います。

たとえば、 Zynq ボード自体をこの目的で使用する場合は、

```
> mkdir /mnt/sd
> mount /dev/mmcblk0p1 /mnt/sd
```

と入力し、ファイルを /mnt/sd/にコピーします。

Windows システムでは、 (micro)SD カードを差し込むと、単一のファイルuImageを持つ単一の “disk”が表示されます。これは、ファイルのコピー先の正しい宛先です。

完了したら、 (Micro)SD カードを正しくアンマウントし、コンピューターから取り外します（

```
> umount /mnt/sd
```

または Windowsの “remove the disk safely” など）。

3.6 boot partitionのファイル

bootを試す前に、 boot partition が次のように設定されていることを確認してください。

boot を成功させるには、 (micro)SD カードの最初の partition （ boot partition）に4つのファイルが存在する必要があります。

- uImage – Linux kernel binary。これは、 Xillinuxの (Micro)SD image をカードに書き込んだ後の boot partition 内の唯一のファイルです。 kernel はボードに依存しません。
- boot.bin –最初の bootloader。このファイルには、初期の processor 初期化と U-boot ユーティリティが含まれており、ボードごとに大きく異なります。
- devicetree.dtb – Linux kernelのハードウェア情報を含む Device Tree Blob ファイル。
- xillydemo.bit –セクション 3.3で生成された PL (FPGA) プログラミングファイル

4

bootをキックオフ

4.1 ジャンパー設定

ボードが (Micro)SD カードから boot を実行するには、ジャンパー設定を変更する必要があります。設定は、以下のボードごとに詳細に説明されています。

4.1.1 Zedboard

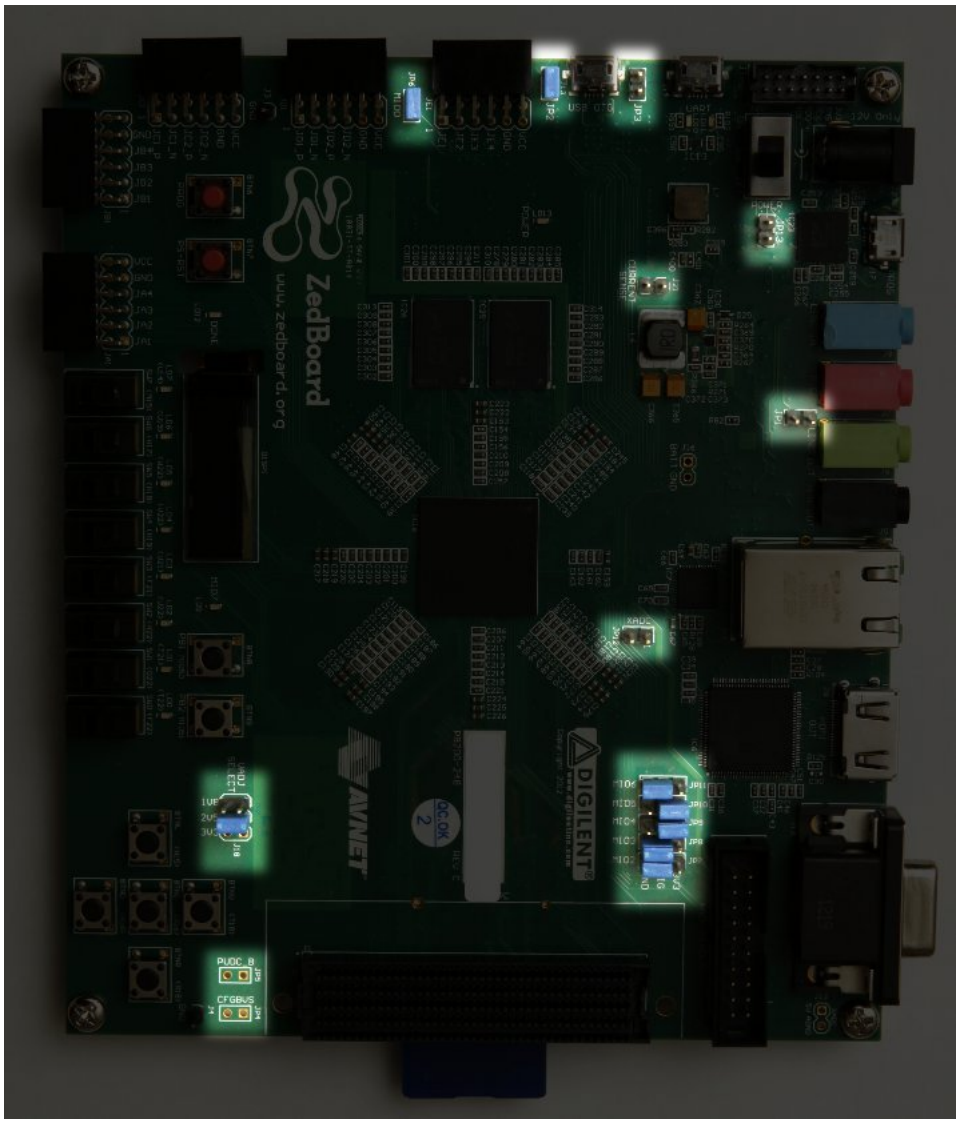
正しい設定は、次のページの画像に示されています。

通常、次のジャンパーの変更が必要です（ただし、ボードの設定が最初から異なる場合があります）。

- JP2 のジャンパーを取り付けて、 5V を USB デバイスに供給します。
- JP10 と JP9 は GND から 3V3 の位置に移動し、その行の他の3つは GNDに接続されたままになります。
- JP6 用のジャンパーを取り付けます（ CES siliconに必要です。 ZedboardのHardware Guideの34ページを参照してください）。

重要:

必要な設定は、 *JP2* がジャンパーされているという点で、 *Zedboard Hardware User Guide* で説明されている設定とは異なります。そのため、ボードに接続されている *USB* デバイス（*USB* キーボードおよびマウス）は *5V* 電源を受け取ります。



Zedboardで強調表示されたジャンパー設定

4.1.2 MicroZed

MicroSD カードから Xillinux の boot を実行するための適切なジャンパー設定は次のとおりです。

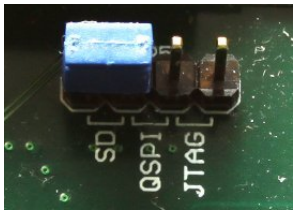
- JP1: 1-2 (GND)
- JP2: 2-3 (VCC)
- JP3: 2-3 (VCC)

デフォルト設定のボードを考えると、 JP2 のみを移動する必要があります。

正しいジャンパー設定を次に示します。

4.1.3 Zybo

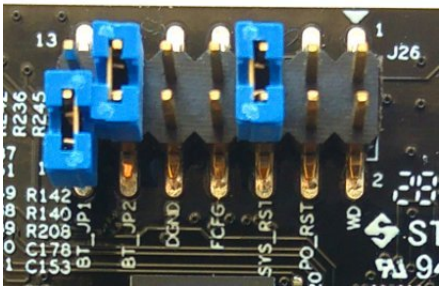
boot mode は、 VGA コネクタの近くにあるジャンパーによって選択されます。このジャンパーは、次の画像に示すように、 “SD”でマークされた2つのピンに設定する必要があります。



他のジャンパーは、目的の動作モードに応じて設定されます。たとえば、電源ジャンパーは、外部 5V ソースまたは UART USB ジャックから電力を取得するように設定できます。どちらも Xillinuxの boot を成功させるには問題ありません。

4.1.4 Z-Turn Lite

Ethernet コネクタ（ J26のラベルが付いている）の横にあるジャンパは boot modeを決定し、次のように設定する必要があります。



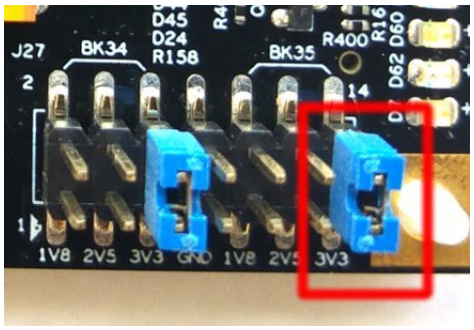
- BT_JP1 ジャンパーは配置しないでください（または図に示すように配置する、つまりピンの1つだけに接続する）
- BT_JP2 ジャンパーを配置する必要があります
- FCFG および PO_RST は配置しないでください

boot の操作とは関係なく、これら2つのジャンパーは重要です。

- SYS_RST ジャンパーを使用すると、ボードの K2 ボタンを押すことで Zynqの processor をリセットできます。
- WD （Watchdog）ジャンパーを使用すると、 processorからオンボード watchdog チップを有効にできます。 Xillinux は、配置されているかどうかに関係なく、 boot を適切に実行します。

DGND ピンペアは、アースに接続された2本のピンです。これらにジャンパーを配置しても効果はありません。

また、 HDMI ビデオ出力を（ Cape IO ボードを介して）使用する場合は、 Zynq デバイスの bank 35 を 3.3V 電圧電源で駆動する必要があります。これは、 J27の BK35 グループの 3V3 ジャンパーを使用して、 MicroSD カードに近いボードのコーナーで確立されます（下の画像を参照）。



4.2 周辺機器の取り付け

ボードには、次の汎用ハードウェアを取り付けることができます。

- Z-Turn Lite IO Cape boardを備えた Z-Turn Lite ： Cape boardの HDMI コネクタへのコンピュータモニター。または、 HDMI/DVI アダプタまたはケーブルを介してボードの HDMI プラグに接続された DVI 入力。
- Zedboard / Zybo：アナログ VGA コネクタへのコンピュータモニター。
- Zybo： HDMI 入力を備えたコンピューターモニター。または、 HDMI/DVI アダプタまたはケーブルを介してボードの HDMI プラグに接続された DVI 入力。
- Zedboard/Zybo/Z-Turn Lite： USB （OTG）コネクタへのマウスとキーボード。

  Zyboには、このための PCのような USB プラグがあります。 Zedboard および Z-Turn Liteでは、これは Zedboard （短い方）に付属の USB メスケーブルを経由します。このケーブルは、 “kit” 構成で購入した場合、 Z-Turn Lite にも同梱されています。

  これらがない場合、システムは boot を正常に実行し、システムの実行中にキーボードとマウスの接続と切断に問題はありません。システムは、任意の時点で接続されているキーボードとマウスを検出して動作します。

  Zedboardでは、この USB ポートが機能するために JP2 をインストールする必要があることに注意してください。
- Ethernet ポートは、一般的なネットワークタスクではオプションです。接続されたネットワークに DHCP serverがある場合、 Linux マシンはネットワークを自動的に構成します。
- UART USB ポートは PCに接続できますが、 Zedboard および Zyboではほとんどの場合これは必要ありません。一部の boot messages がそこに送信され、 boot が完了すると、このインターフェイスで shell prompt が発行されます。

  これは、 boot中の障害のデバッグ、または PC モニターまたはキーボードのいずれかが欠落しているか正しく機能しない場合に役立ちます。

  Z-Turn Liteの場合、ボードの 3.3V UART 信号とのインターフェースには外部 USB アダプターが必要です。 Z-Turn Liteに関して Xillybusの web page で説明されているように、このアダプタはボードに含まれている可能性があります。

4.3 ボードの電源を入れる

4.3.1 初期診断

上記のビルド手順に従っている場合、 boot 中に障害が発生することはまれです。一般的な理由は次のとおりです。

- ジャンパー設定が正しくありません（ 4.1項を参照）。
- Sandisk製ではない (Micro)SD カードを使用する。カードが正常に機能しているように見えても、散在するデータの破損は簡単に見落とされますが、まったく別の理由でエラーが発生するように見えます。
- (Micro)SD image のカードへの書き込みが不完全または誤っている
- ボードの QSPI flashから U-boot によってロードされた古くて不十分な環境変数。 4.3.3項を参照してください。
- 通常、システムを最初に構築しようとしたときにシステムを微調整しようとしたため、指示に従わない。

正しい UART 設定は、 115200 baud、 8 data bits、 1 stop bitsであり、 flow controlはありません。ボードの電源を入れてから4秒以内にテキストが表示されるはずです。これを行うための唯一の要件は、 boot.bin ファイルが (Micro)SD カードの最初の（FAT32） partition にあることです。

ボードの電源を入れても何も起こらない場合：

- ボードのタイプに一致する正しい boot.bin が boot partition kit からコピーされていることを確認します。キットのファイル名は、キットを使用するボードを示しています。
- UART からコンピューターへのリンクが正しく機能することを確認します。 QSPI flash または（おそらく）ボードに付属している SD カードのサンプル Linux を使用している可能性があります。 UART terminal アプリケーション用の host としての Linuxは、一部の UART/USB コンバーターでは正しく動作しない可能性があるため、 Windows で Tera Term を試すことが唯一のオプションである可能性があることに注意してください。

U-boot が consoleでメッセージを出力するが、 boot プロセスが失敗する場合は、段落 4.3.5のトランスクリプトと比較すると役立つ場合があります。このセクションの残りの部分には、何が問題なのかを理解するための関連情報も含まれている場合があります。

4.3.2 boot プロセスが完了したとき

boot プロセスの最後に、 shell prompt が UART consoleに提供され、手動でログインする必要はありません。それでも、 root userのパスワードは何も設定されていないため、 rootとしてログインする必要がある場合でも、パスワードは必要ありません。

重要:

ボードに付属の *(Micro)SD* カードの *Linux* サンプルとは異なり、 *Xillinuxの root file system* は *(Micro)SD* カードに永続的に常駐し、システムの起動中に書き込まれます。 *Linux* システムは、ボードの電源を切る前に適切にシャットダウンして、システムを安定させる必要があります。これは、他の*PC*コンピュータと同様に、通常、突然の電力損失後に適切な回復が見られる場合でも同様です。

Z-Turn Lite、 Zedboard 、および Zyboに関する注記：

- shell prompt で “startx” と入力して、 LXDE graphical desktopを起動します。 desktop の初期化には15∼30秒かかります。何も起こらないように見える場合は、 OLED ディスプレイのアクティビティメーターを監視すると、何かが起こっているかどうかを確認するのに役立ちます（ Zedboard のみがこの OLED ディスプレイを備えています）。
- logic fabric がロードされてから Linux kernel が起動するまで、背景が白の Xillybus ロゴスクリーンセーバーが画面に表示されます。また、オペレーティングシステムが画面を “blank” モードにしたときも表示されます。これは、システムがアイドル状態のときの通常の状態、または X-Windows システムがグラフィックモードを操作しようとしたときです。
- **青い**背景の Xillybus スクリーンセーバー、または画面上のランダムな青いストライプは、グラフィックインターフェイスがデータ不足に苦しんでいることを示しています。明らかな理由がわからない限り、これが発生することは決して予想されておらず、報告する必要があります。

4.3.3 U-boot 環境変数

Xillinux は、 boot プロセス中に xillydemo.bit、 kernel image 、および device tree をロードするために U-boot に依存しています。このユーティリティは、さまざまな boot 構成を提供し、設定の実験と変更を可能にするシンプルなコマンドラインインターフェイスを備えています。 U-boot の起動直後に UART console で任意の文字を入力すると、

U-bootの shell に到達します。

```
U-Boot 2013.07 (Mar 15 2014 - 22:59:21)

Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL129P_64K/S25FL128S_64K with page size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  1
```

U-boot は、常に QSPI flashから保存された環境変数を取得しようとします。“bad CRC” と表示されている warning は、有効なデータが見つからなかったことを示しているため、 U-boot はハードコードされたデフォルト環境に戻りました。 Xillinuxの U-boot にはすべての環境変数が正しく設定されているため、これは (Micro)SD カードの Xillinux の boot のエラーではありません。

重要:

システムが *(Micro)SD* カードから *boot* を実行する場合でも、環境変数はボード自体の *QSPI flash* から取得されることに注意してください。 *QSPI flash* に別の *boot* シナリオに一致する環境変数が含まれている場合、 *U-boot* は、不適切な変数に依存して、 *boot* プロセスに失敗する可能性があります。

キーが1秒間押されない場合、 U-boot はその環境変数（ QSPI flashからロードされた変数、またはハードコードされたデフォルト設定）に従って続行します。より正確には、 “bootcmd” 変数のコンテンツを実行します。これはデフォルトで “run $modeboot” と表示されます。次に、 “modeboot”は、 U-boot がロードされた場所に応じて、 U-bootによって動的に設定されるため、通常の Xillinux bootでは “sdboot” と表示されます。 “sdboot” 変数には、 Xillinuxの boot プロセスを実行するための一連のコマンドが含まれています。

U-bootの command-line shell には “help” コマンドがあり、すべてのコマンドとその意味が一覧表示されます。いくつかの便利なコマンドは

- help *command* – *command* のヘルプを表示

- env print –すべての環境変数の現在の値を出力します
- env set –特定の環境変数の値を設定します
- env default -a –すべての環境変数をハードコードされたデフォルトに設定します。
- saveenv –現在の環境変数を（ MicroSD/SD カードではなく） QSPI flash に保存します。

特に、リストの最後の2つのコマンドは、 U-boot が boot プロセスに失敗した場合に重要です。 “bad CRC, using default environment” を示す warning が U-bootによって発行されて**いない**場合、格納されている変数に依存しています。デフォルトの変数（ Xillinuxに適している）を使用するには、次のようにします。

```
xillinux-uboot> env default -a
## Resetting to default environment
xillinux-uboot> saveenv
Saving Environment to SPI Flash...
SF: Detected S25FL129P_64K/S25FL128S_64K with page size 64 KiB, total 16 MiB
Erasing SPI flash...Writing to SPI flash...done
```

保存された環境変数に望ましい変更があった場合は、もちろんそれらも消去されます。

4.3.4 カスタム Ethernet MAC アドレスの設定

Linux は、特定の環境変数に応じて、 U-bootによって設定されるハードウェア上で検出された Ethernet MAC アドレスに依存します。環境変数は QSPI flashに格納されているため、 MAC アドレスは永続的にハードウェアにバインドされます。したがって、特定の (Micro)SD カードが異なるボードで使用されている場合でも、各ボードは独自の MAC アドレスを保持します。

たとえば、 USB UART consoleを使用する U-bootの shell では、次のようになります。

```
xillinux-uboot> set ethaddr 00:11:22:33:44:55
xillinux-uboot> saveenv
Saving Environment to SPI Flash...
Erasing SPI flash...Writing to SPI flash...done
```

後で、ボードの電源をリサイクルし、 Linux に boot を自動的に実行させた後：

```
root@localhost:~# ifconfig
eth1      Link encap:Ethernet  HWaddr 00:11:22:33:44:55
          inet addr:10.1.1.164  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::211:22ff:fe33:4455/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:50 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2720 (2.7 KB)  TX bytes:9230 (9.2 KB)
          Interrupt:54 Base address:0xb000
```

（ IP アドレスは、上記の場合、 DHCP server によって指定されました）

4.3.5 boot中のサンプルトランスクリプト

参考までに、 boot 中の典型的な UART トランスクリプトを以下に示します。 Zedboardの例を示していますが、ボード間の違いはわずかです。 boot プロセスが失敗した場合、エラー メッセージに、おそらくどの段階で問題が発生したか、およびその理由が示されます。

```
U-Boot 2013.07 (Aug 10 2014 - 11:28:31)

Zynq PS_VERSION = 0
Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL256S_64K with page size 64 KiB, total 32 MiB
In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  1  0
Device: zynq_sdhci
Manufacturer ID: 3
OEM: 5344
Name: SL08G
Tran Speed: 50000000
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 7.4 GiB
Bus Width: 4-bit
Booting Xillinux...
reading xillydemo.bit
4045675 bytes read in 295 ms (13.1 MiB/s)
  design filename = "xillydemo.ncd;HW_TIMEOUT=FALSE;UserID=0xFFFFFFFF"
  part number = "7z020clg484"
  date = "2014/03/11"
  time = "22:22:00"
  bytes in bitstream = 4045564
zynq_load: Align buffer at 10006f to 100080(swap 1)
reading uImage
4487928 bytes read in 326 ms (13.1 MiB/s)
reading devicetree.dtb
9531 bytes read in 16 ms (581.1 KiB/s)
## Booting kernel from Legacy Image at 03000000 ...
   Image Name:   Linux-4.4.30-xillinux-2.0
   Image Type:   ARM Linux Kernel Image (uncompressed)
```

```
   Data Size:    4487864 Bytes = 4.3 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 02a00000
   Booting using the fdt blob at 0x2a00000
   Loading Kernel Image ... OK
   Loading Device Tree to 1fb4e000, end 1fb5353a ... OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.30-xillinux-2.0 (eli@ocho.localdomain) (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-40) ) #1 SMP PREEMPT Tue D
[    0.000000] CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] Machine model: Xilinx Zynq
[    0.000000] bootconsole [earlycon0] enabled
[    0.000000] cma: Reserved 16 MiB at 0x1e800000
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] PERCPU: Embedded 12 pages/cpu @dfb36000 s18880 r8192 d22080 u49152
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 129920
[    0.000000] Kernel command line: console=ttyPS0,115200n8 console=tty0 consoleblank=0 root=/dev/mmcblk0p2 rw rootwait earlyprintk
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Memory: 493232K/524288K available (6155K kernel code, 294K rwdata, 2192K rodata, 312K init, 472K bss, 14672K reserved, 16384K cma-reser
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff800000   ( 496 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc082f0c4   (8349 kB)
[    0.000000]       .init : 0xc0830000 - 0xc087e000   ( 312 kB)
[    0.000000]       .data : 0xc087e000 - 0xc08c7840   ( 295 kB)
[    0.000000]        .bss : 0xc08c7840 - 0xc093da38   ( 473 kB)
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000]  Build-time adjustment of leaf fanout to 32.
[    0.000000]  RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
[    0.000000] RCU: Adjusting geometry for rcu_fanout_leaf=32, nr_cpu_ids=2
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] slcr mapped to e0800000
[    0.000000] L2C: platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C: DT/platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C-310 erratum 769419 enabled
[    0.000000] L2C-310 enabling early BRESP for Cortex-A9
[    0.000000] L2C-310 full line of zeros enabled for Cortex-A9
[    0.000000] L2C-310 ID prefetch enabled, offset 1 lines
[    0.000000] L2C-310 dynamic clock gating enabled, standby mode enabled
[    0.000000] L2C-310 cache controller enabled, 8 ways, 512 kB
[    0.000000] L2C-310: CACHE_ID 0x410000c8, AUX_CTRL 0x76760001
[    0.000000] zynq_clock_init: clkc starts at e0800100
[    0.000000] Zynq clock init
[    0.000000] clocksource: ttc_clocksource: mask: 0xffff max_cycles: 0xffff, max_idle_ns: 537538477 ns
[    0.000018] sched_clock: 16 bits at 54kHz, resolution 18432ns, wraps every 603975816ns
[    0.007925] ps7-ttc #0 at e0808000, irq=17
[    0.012173] sched_clock: 64 bits at 333MHz, resolution 3ns, wraps every 4398046511103ns
[    0.020052] clocksource: arm_global_timer: mask: 0xffffffffffffffff max_cycles: 0x4ce07af025, max_idle_ns: 440795209040 ns
[    0.031309] Console: colour dummy device 80x30
[    0.035629] console [tty0] enabled
[    0.039067] bootconsole [earlycon0] disabled
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.30-xillinux-2.0 (eli@ocho.localdomain) (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-40) ) #1 SMP PREEMPT Tue D
[    0.000000] CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] Machine model: Xilinx Zynq
[    0.000000] bootconsole [earlycon0] enabled
[    0.000000] cma: Reserved 16 MiB at 0x1e800000
```

```
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] PERCPU: Embedded 12 pages/cpu @dfb36000 s18880 r8192 d22080 u49152
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 129920
[    0.000000] Kernel command line: console=ttyPS0,115200n8 console=tty0 consoleblank=0 root=/dev/mmcblk0p2 rw rootwait earlyprintk
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Memory: 493232K/524288K available (6155K kernel code, 294K rwdata, 2192K rodata, 312K init, 472K bss, 14672K reserved, 16384K cma-rese
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff800000   ( 496 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc082f0c4   (8349 kB)
[    0.000000]       .init : 0xc0830000 - 0xc087e000   ( 312 kB)
[    0.000000]       .data : 0xc087e000 - 0xc08c7840   ( 295 kB)
[    0.000000]        .bss : 0xc08c7840 - 0xc093da38   ( 473 kB)
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000]  Build-time adjustment of leaf fanout to 32.
[    0.000000]  RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
[    0.000000] RCU: Adjusting geometry for rcu_fanout_leaf=32, nr_cpu_ids=2
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] slcr mapped to e0800000
[    0.000000] L2C: platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C: DT/platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C-310 erratum 769419 enabled
[    0.000000] L2C-310 enabling early BRESP for Cortex-A9
[    0.000000] L2C-310 full line of zeros enabled for Cortex-A9
[    0.000000] L2C-310 ID prefetch enabled, offset 1 lines
[    0.000000] L2C-310 dynamic clock gating enabled, standby mode enabled
[    0.000000] L2C-310 cache controller enabled, 8 ways, 512 kB
[    0.000000] L2C-310: CACHE_ID 0x410000c8, AUX_CTRL 0x76760001
[    0.000000] zynq_clock_init: clkc starts at e0800100
[    0.000000] Zynq clock init
[    0.000000] clocksource: ttc_clocksource: mask: 0xffff max_cycles: 0xffff, max_idle_ns: 537538477 ns
[    0.000018] sched_clock: 16 bits at 54kHz, resolution 18432ns, wraps every 603975816ns
[    0.007925] ps7-ttc #0 at e0808000, irq=17
[    0.012173] sched_clock: 64 bits at 333MHz, resolution 3ns, wraps every 4398046511103ns
[    0.020052] clocksource: arm_global_timer: mask: 0xffffffffffffffff max_cycles: 0x4ce07af025, max_idle_ns: 440795209040 ns
[    0.031309] Console: colour dummy device 80x30
[    0.035629] console [tty0] enabled
[    0.039067] bootconsole [earlycon0] disabled
[    0.043389] Calibrating delay loop... 1332.01 BogoMIPS (lpj=6660096)
[    0.130990] pid_max: default: 32768 minimum: 301
[    0.131116] Security Framework initialized
[    0.131135] Yama: becoming mindful.
[    0.131211] AppArmor: AppArmor initialized
[    0.131270] Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.131295] Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.132028] Initializing cgroup subsys io
[    0.132059] Initializing cgroup subsys memory
[    0.132104] Initializing cgroup subsys devices
[    0.132133] Initializing cgroup subsys freezer
[    0.132156] Initializing cgroup subsys net_cls
[    0.132177] Initializing cgroup subsys perf_event
[    0.132200] Initializing cgroup subsys net_prio
[    0.132222] Initializing cgroup subsys pids
[    0.132274] CPU: Testing write buffer coherency: ok
[    0.132537] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.132602] Setting up static identity map for 0x82c0 - 0x82f4
[    0.310974] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.311078] Brought up 2 CPUs
[    0.311115] SMP: Total of 2 processors activated (2664.03 BogoMIPS).
[    0.311133] CPU: All CPU(s) started in SVC mode.
[    0.312116] devtmpfs: initialized
[    0.314713] evm: security.selinux
[    0.314734] evm: security.SMACK64
[    0.314748] evm: security.SMACK64EXEC
[    0.314761] evm: security.SMACK64TRANSMUTE
[    0.314775] evm: security.SMACK64MMAP
[    0.314804] evm: security.ima
[    0.314824] evm: security.capability
[    0.315239] VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 4
[    0.315600] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
```

```
[    0.316774] pinctrl core: initialized pinctrl subsystem
[    0.318050] NET: Registered protocol family 16
[    0.320031] DMA: preallocated 256 KiB pool for atomic coherent allocations
[    0.323590] zynq_gpio e000a000.ps7-gpio: This is the Xillinux-1.3 compliant legacy GPIO driver.
[    0.324184] zynq_gpio e000a000.ps7-gpio: gpio at 0xe000a000 mapped to 0xe0814000
[    0.329041] hw-breakpoint: found 5 (+1 reserved) breakpoint and 1 watchpoint registers.
[    0.329099] hw-breakpoint: maximum watchpoint size is 4 bytes.
[    0.375012] vgaarb: loaded
[    0.377592] SCSI subsystem initialized
[    0.378094] usbcore: registered new interface driver usbfs
[    0.378220] usbcore: registered new interface driver hub
[    0.378369] usbcore: registered new device driver usb
[    0.378741] media: Linux media interface: v0.10
[    0.378849] Linux video capture interface: v2.00
[    0.379225] pps_core: LinuxPPS API ver. 1 registered
[    0.379263] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>
[    0.379346] PTP clock support registered
[    0.379675] EDAC MC: Ver: 3.0.0
[    0.383475] NetLabel: Initializing
[    0.383515] NetLabel:  domain hash size = 128
[    0.383538] NetLabel:  protocols = UNLABELED CIPSOv4
[    0.383614] NetLabel:  unlabeled traffic allowed by default
[    0.384003] clocksource: Switched to clocksource arm_global_timer
[    0.384740] AppArmor: AppArmor Filesystem Enabled
[    0.399611] NET: Registered protocol family 2
[    0.400379] TCP established hash table entries: 4096 (order: 2, 16384 bytes)
[    0.400470] TCP bind hash table entries: 4096 (order: 3, 32768 bytes)
[    0.400579] TCP: Hash tables configured (established 4096 bind 4096)
[    0.400652] UDP hash table entries: 256 (order: 1, 8192 bytes)
[    0.400701] UDP-Lite hash table entries: 256 (order: 1, 8192 bytes)
[    0.400961] NET: Registered protocol family 1
[    0.402023] RPC: Registered named UNIX socket transport module.
[    0.402065] RPC: Registered udp transport module.
[    0.402090] RPC: Registered tcp transport module.
[    0.402114] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.402789] hw perfevents: enabled with armv7_cortex_a9 PMU driver, 7 counters available
[    0.404341] futex hash table entries: 512 (order: 3, 32768 bytes)
[    0.404505] audit: initializing netlink subsys (disabled)
[    0.404585] audit: type=2000 audit(0.379:1): initialized
[    0.405111] Initialise system trusted keyring
[    0.405881] VFS: Disk quotas dquot_6.6.0
[    0.405984] VFS: Dquot-cache hash table entries: 1024 (order 0, 4096 bytes)
[    0.406373] squashfs: version 4.0 (2009/01/31) Phillip Lougher
[    0.407215] NFS: Registering the id_resolver key type
[    0.407288] Key type id_resolver registered
[    0.407315] Key type id_legacy registered
[    0.407361] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[    0.407468] jffs2: version 2.2. (NAND) (SUMMARY)  Â© 2001-2006 Red Hat, Inc.
[    0.407949] Allocating IMA MOK and blacklist keyrings.
[    0.409634] Key type asymmetric registered
[    0.409681] Asymmetric key parser 'x509' registered
[    0.409832] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 248)
[    0.410040] io scheduler noop registered
[    0.410079] io scheduler deadline registered (default)
[    0.410137] io scheduler cfq registered
[    0.440104] Console: switching to colour frame buffer device 128x48
[    0.468985] xuartps e0001000.serial: clock name 'aper_clk' is deprecated.
[    0.469289] xuartps e0001000.serial: clock name 'ref_clk' is deprecated.
[    0.469614] e0001000.serial: ttyPS0 at MMIO 0xe0001000 (irq = 158, base_baud = 3125000) is a xuartps
[    1.278046] console [ttyPS0] enabled
[    1.282451] xdevcfg f8007000.ps7-dev-cfg: ioremap 0xf8007000 to e0872000
[    1.305388] brd: module loaded
[    1.315816] loop: module loaded
[    1.337113] libphy: Fixed MDIO Bus: probed
[    1.343139] libphy: XEMACPS mii bus: probed
[    1.348856] xemacps e000b000.ps7-ethernet: pdev->id -1, baseaddr 0xe000b000, irq 31
[    1.357688] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
[    1.364433] ehci-pci: EHCI PCI platform driver
[    1.369044] ehci-platform: EHCI generic platform driver
[    1.381383] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
[    1.394522] ohci-pci: OHCI PCI platform driver
[    1.405966] ohci-platform: OHCI generic platform driver
[    1.418231] uhci_hcd: USB Universal Host Controller Interface driver
[    1.431756] usbcore: registered new interface driver usb-storage
[    1.445354] mousedev: PS/2 mouse device common for all mice
[    1.458696] i2c /dev entries driver
```

```
[    1.470432] device-mapper: uevent: version 1.0.3
[    1.482312] device-mapper: ioctl: 4.34.0-ioctl (2015-10-28) initialised: dm-devel@redhat.com
[    1.497966] sdhci: Secure Digital Host Controller Interface driver
[    1.511276] sdhci: Copyright(c) Pierre Ossman
[    1.522726] sdhci-pltfm: SDHCI platform and OF driver helper
[    1.537085] sdhci-arasan e0100000.ps7-sdio: No vmmc regulator found
[    1.550571] sdhci-arasan e0100000.ps7-sdio: No vqmmc regulator found
[    1.564036] mmc0: Invalid maximum block size, assuming 512 bytes
[    1.614085] mmc0: SDHCI controller on e0100000.ps7-sdio [e0100000.ps7-sdio] using ADMA
[    1.629895] ledtrig-cpu: registered to indicate activity on CPUs
[    1.644279] Key type dns_resolver registered
[    1.656171] Registering SWP/SWPB emulation handler
[    1.666380] mmc0: new high speed SDHC card at address aaaa
[    1.677100] mmcblk0: mmc0:aaaa SL08G 7.40 GiB
[    1.678486]  mmcblk0: p1 p2
[    1.703249] registered taskstats version 1
[    1.714453] Loading compiled-in X.509 certificates
[    1.727453] Key type encrypted registered
[    1.738464] AppArmor: AppArmor sha1 policy hashing enabled
[    1.750995] ima: No TPM chip found, activating TPM-bypass!
[    1.763635] evm: HMAC attrs: 0x1
[    1.774166] hctosys: unable to open rtc device (rtc0)
[    1.791487] md: Waiting for all devices to be available before autodetect
[    1.805427] md: If you don't use raid, use raid=noautodetect
[    1.819245] md: Autodetecting RAID arrays.
[    1.830359] md: Scanned 0 and added 0 devices.
[    1.841633] md: autorun ...
[    1.851105] md: ... autorun DONE.
[    1.861835] EXT4-fs (mmcblk0p2): couldn't mount as ext3 due to feature incompatibilities
[    1.877515] EXT4-fs (mmcblk0p2): couldn't mount as ext2 due to feature incompatibilities
[    1.906578] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[    1.921636] VFS: Mounted root (ext4 filesystem) on device 179:2.
[    1.942290] devtmpfs: mounted
[    1.952285] Freeing unused kernel memory: 312K (c0830000 - c087e000)
[    2.204187] systemd[1]: System time before build time, advancing clock.
[    2.323264] NET: Registered protocol family 10
[    2.372338] random: systemd: uninitialized urandom read (16 bytes read, 6 bits of entropy available)
[    2.390906] random: systemd: uninitialized urandom read (16 bytes read, 6 bits of entropy available)
[    2.414805] systemd[1]: systemd 229 running in system mode. (+PAM +AUDIT +SELINUX +IMA +APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNU
[    2.447961] systemd[1]: Detected architecture arm.
[    2.491022] systemd[1]: Set hostname to <localhost.localdomain>.
```

そしてそれは続き、 systemd の初期化を伴います。 30秒以内に、 shell prompt が次のように表示されます。この最後の出力の前に、おそらく数秒の休止があります。

```
Ubuntu 16.04 LTS localhost.localdomain ttyPS0

localhost login: root (automatic login)

Last login: Thu Feb 11 16:28:21 UTC 2016 on ttyPS0
Welcome to the Xillinux-2.0 distribution for Xilinx Zynq.

You may communicate data with standard FPGA FIFOs in the logic fabric by
writing to or reading from the /dev/xillybus_* device files. Additional
pipe files of that sort can be set up with a custom Xillybus IP core.

For more information: http://www.xillybus.com.

To start a graphical X-Windows session, type "startx" at shell prompt.
```

```
root@localhost:~#
```

4.4 最初の bootの直後に行う

4.4.1 file systemのサイズを変更します

root file systemの image は小型に保たれているため、デバイスへの書き込みは可能な限り高速です。一方、 (Micro)SD カードの全容量を使用しない理由はありません。

重要:

*file system*のサイズを変更しようとすると、 *(Micro)SD* カードのコンテンツ全体が消去されるという重大なリスクがあります。したがって、これをできるだけ早く行うことをお勧めしますが、このような事故のコストは、 *(Micro)SD* カードの初期化を繰り返すことです（ *image* の書き込みと *boot partition*への入力）。

開始点は通常、次のとおりです。

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       3.4G  2.8G  388M  89% /
devtmpfs        241M     0  241M   0% /dev
tmpfs           249M   72K  249M   1% /dev/shm
tmpfs           249M  7.2M  242M   3% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs            50M  4.0K   50M   1% /run/user/0
```

したがって、 root filesystem は 2.8 GBであり、 388 MB freeを備えています。

最初の段階は、 (Micro)SD カードを再パーティション化することです。 shell promptで、

```
# fdisk /dev/mmcblk0
```

と入力してから、次のように入力します（以下のセッショントランスクリプトも参照してください）。

- d [ENTER] – partition を削除
- 2 [ENTER] – partition 番号2を選択
- n [ENTER] –新しい partition を作成します
- [ENTER] を4回押して、デフォルトを受け入れます。 primary partition、番号2は、可能な限り低い sector で始まり、可能な限り高い sector で終わります。
- w[ENTER]-保存して終了します。

このシーケンスの途中で問題が発生した場合は、 CTRL-C（または q [ENTER]）を押して、変更を保存せずに fdisk を終了します。 (Micro)SD カードでは、最後の手順まで何も変更されません。

典型的なセッションは次のようになります。 sector の番号は異なる場合があることに注意してください。

```
# fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): d
Partition number (1,2, default 2): 2

Partition 2 has been deleted.

Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (2-4, default 2):
First sector (32768-15523839, default 32768):
Last sector, +sectors or +size{K,M,G,T,P} (32768-15523839, default 15523839):

Created a new partition 2 of type 'Linux' and of size 7.4 GiB.
```

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Re-reading the partition table failed.: Device or resource busy

The kernel still uses the old table. The new table will be used at
the next reboot or after you run partprobe(8) or kpartx(8).
```

システムに表示されるデフォルトの最初の sector が上記のものと異なる場合は、ここに示されているものではなく、システムのデフォルトを選択してください。

このシーケンスで、 fdiskのデフォルトから逸脱することが理にかなっている唯一の場所は、 file system を可能な最大値よりも小さくするための最後の sectorです（ただし、これを行う必要はありません）。

下部の warning にあるように、 Linuxの partition table のビューは更新されません。提案に従って、次のように入力します。

```
# partprobe
```

これにより、 consoleへの出力は生成されません。 partition 2の変更を kernel に通知できないと文句を言う場合は、 partprobe がそれを見つけられなかったことが原因である可能性があります。言い換えれば、 root partition は partition table がそうあるべきだと言っている場所ではありません。おそらく、 fdiskで問題が発生したため、修正する必要があります。そうしないと、 Linux は boot を再度実行できなくなります。

partprobe がサイレントの場合、 partition table は問題ありませんが、 file system のサイズはまだ変更されていません。サイズを変更する余地があるだけです。したがって、 shell promptで、次の応答が期待される

```
# resize2fs /dev/mmcblk0p2
```

と入力します。

```
resize2fs 1.42.13 (17-May-2015)
Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mmcblk0p2 is now 1936384 (4k) blocks long.
```

block count は partitionのサイズによって異なるため、異なる場合があります。

ユーティリティが言うように、サイズ変更はアクティブに使用されている file system で行われます。途中で電源が切れない限り安全です。

結果はすぐに有効になります。再起動する必要はありません。

8 GB (Micro)SD カードを使用した一般的なセッション：

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.1G  2.8G  4.0G  42% /
devtmpfs        241M     0  241M   0% /dev
tmpfs           249M   72K  249M   1% /dev/shm
tmpfs           249M  7.2M  242M   3% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs            50M  4.0K   50M   1% /run/user/0
```

“df -h” ユーティリティによって指定されるサイズは 1 GiB = 2^{30} バイトであり、 10^9 バイトの Gigabyte よりも 7.1% 大きいことに注意してください。そのため、 8 GB カードは上記の 7.1 GiB として表示されます。

4.4.2 リモート SSH アクセスを許可する

root password はデフォルトではなしであり、すべてのユーザーがパスワードなしで root としてログインできます。その結果、 ssh は rootへのログインを拒否します。

これを修正するには、 shell promptで次のコマンドを使用して root password を設定します。

```
# passwd
```

4.4.3 ロケール定義のCompilation （必要な場合）

特定の状況では、アプリケーションソフトウェアは、 locale settingsに応じて文字を表示する方法に関する知識を必要とします。最も一般的な理由は、 ssh をボードに接続する場合です。これは、 ssh が shell sessionのセットアップ時に clientの locale settings を serverの環境にコピーするためです。

これは warning messages のようにつながる可能性があります

```
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8)
```

このようなエラーメッセージが表示された場合、どの locale が欠落しているかは明らかです。エラーが発生する前にこれを解決するには、どの locales が必要かを確認してください。

```
# locale
LANG=en_US.UTF-8
LANGUAGE=
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

利用可能な localesと比較してください。

```
# locale -a
C
C.UTF-8
POSIX
```

この例では、どの locale が欠落しているかが非常に明らかなので、追加してみましょう。

```
# locale-gen en_US.UTF-8
Generating locales...
  en_US.UTF-8... done
```

必要な locale は、 ssh 接続が行われているコンピューターによって異なることに注意してください。スムーズな ssh セッションを実現するには、世界中のさまざまな場所のユーザーがボードにさまざまな locales をインストールする必要があります。 UART ポートの shell は、デフォルトで含まれている POSIX localeに基づいています。

en_US.UTF-8 はかなりユビキタスであるため、すでにディストリビューションにインストールされています（上記のセッション例では反対のことが示されていますが）。

4.5 desktopの使用

Xillinux desktop （ Z-Turn Lite、 Zedboard 、および Zybo）は、他の Lubuntu desktopとまったく同じです。 (Micro)SD カードのデータ帯域幅が比較的低いため、アプリケーションのロードがやや遅くなる場合がありますが、 desktop 自体はかなり応答性が高くなります。

追加のパッケージは、他の Ubuntu ディストリビューションと同様に “apt-get” でインストールできます。

Ubuntu オペレーティングシステム全体を apt でアップグレードすることは不可能であり、失敗し、システムが boot を再度実行する可能性がなくなります。

4.6 シャットダウン/再起動

システムの電源を切るには、デスクトップの右下のアイコン（使用可能な場合）を選択し、[“Shutdown”]をクリックするか、その他のオプションを適切に選択します。 “Lock Screen” は何もしません。

または、 shell promptで次のように入力します。

```
# halt
```

“System Halted” を示すテキストメッセージが UART console（および存在する場合は画面）に表示されたら、ボードの電源をオフにしても安全です。

bitstream を FPGA （PL）パーツにリロードすることを含む rebootの場合、デスクトップメニューで reboot オプションを選択するか、次のように入力します。

```
# reboot
```

これは必ずしも外部ハードウェアコンポーネント（サウンドチップなど）をリセットするわけではないことに注意してください。

4.7 ここから何をすべきか

Zynq ボードは、あらゆる目的で Linux を実行するコンピューターになりました。 Xillybus IP core を介して logic fabric と対話するための基本的な手順は、

Getting started with Xillybus on a Linux hostにあります。 Xillybus 用の driver はすでに Xillinux ディストリビューションにインストールされているため、インストールに関するガイドの部分はスキップできます。

段落 5.1 は、アプリケーション固有の logic を Linux オペレーティングシステムと統合することを示しています。

あるいは、 Verilog / VHDL プログラミングを避けたいユーザーは、 The guide to Xillybus Block Design Flow for non-HDL usersで概説されているように Block Design Flowを選択することができます。ただし、この方法では Verilog / VHDLと比較してXillybusの機能のセットが弱いため、目的のアプリケーションが block design に自然に適合しない限り（たとえば、 Vivadoの High Level Synthesis, HLSと組み合わせて）、この方法は避ける必要があります。

Xillinux には gcc compiler と GNU makeが含まれているため、通常のコンピュータプログラムの compilation をボードの processorsで直接実行できることに注意してください。 apt-get を使用して、ディストリビューションに追加のパッケージを追加することもできます。

5

変更を加える

5.1 カスタム logicとの統合

この部分は、 Verilog / VHDLでの作業に関連しています。 Block Design Flow を選択したユーザーは、 The guide to Xillybus Block Design Flow for non-HDL usersの指示に従う必要があります。

Xillinux ディストリビューションは、アプリケーション logicと簡単に統合できるように設定されています。データ ソースとデータ コンシューマーを接続するためのフロント エンドは、 xillydemo.v または xillydemo.vhd ファイルです (優先言語によって異なります)。 boot partition kit 内の他のすべての HDL ファイルは、 Linux host と logic fabric間のデータ転送として Xillybus IP core を使用する目的で無視できます。

カスタム logic designs を含む追加の HDL ファイルを、段落 3.3に示されているプロジェクトに追加して、最初に行ったのと同じ方法で再構築することができます。更新された logicを使用してシステムの boot を実行するには、新しい xillydemo.bit を (Micro)SD カードの boot partitionにコピーして、既存のカードを上書きします。 3.5項に示すように、 xillydemo.bit を boot partitionにコピーするために Zynq ボード自体を使用できることに注意してください。

最初の配布展開の他の手順を繰り返す必要がないため、 logic の開発サイクルはかなり迅速かつ簡単です。

JTAG を介した PL パーツのプログラミングはサポートされていません。

Xillybus IP core をカスタム application logicに接続する場合は、 FIFOsを介してのみ Xillybus IP core と対話し、少なくとも最初の段階では、 FIFOの動作を logicで模倣しようとしないことを強くお勧めします。

これの例外は、 Xillybus を block RAM または registersに接続する場合です。この場合、 xillydemo モジュールに示されている方法に従う必要があります。

xillydemo モジュールでは、 FIFOs を使用して、 host から到着し、 host に戻るデータの loopback を実行します。 FIFOsの両側が Xillybus IP coreに接続されているため、 core は独自のデータ ソースおよびデータ コンシューマーとして機能します。

より有用なシナリオでは、 FIFOの一方の端のみが Xillybus IP coreに接続され、もう一方の端はアプリケーション データ ソースまたはデータ コンシューマーに接続されます。

xillydemo モジュールで使用される FIFOs は、両側が Xillybusのメイン clockによって駆動されるため、両側に共通の clock を 1 つだけ受け入れます。実際のアプリケーションでは、 bus clock以外の clock でデータ ソースとデータ コンシューマーを駆動できるように、読み取りと書き込み用に個別の clocks を持つ FIFOs に置き換えることが望ましい場合があります。これにより、 FIFOs は仲介者としてだけでなく、適切な clock domain crossingに対しても機能します。

Xillybus IP core は、 FPGA から hostまでの streams に対して、（ First Word Fall Throughではなく）プレーンな FIFO を想定していることに注意してください。

次のドキュメントは、カスタム logicの統合に関連しています。

- logic design用の API ： Xillybus FPGA designer's guide
- Linux hostの基本概念： Getting started with Xillybus on a Linux host
- プログラミングアプリケーション： Xillybus host application programming guide for Linux
- カスタム Xillybus IP coreのリクエスト： The guide to defining a custom Xillybus IP core

5.2 他のボードを使用する

Z-Turn Lite、 Zedboard、 MicroZed 、または Zybo以外のボードで Xillinux を実行する前に、特定の変更が必要になる場合があります。

ただし、手順が難しいため、 Xillinux を他のハードウェアに適合させることはお勧めしません。経験によれば、 Xillinux を適応させる目的が、 Xillybus IP coreを使用すること以外である場合、最初から始める方が簡単です。

これは注意を払うべき問題の部分的なリストです。

- 購入したボードには、参照として XML ファイルが必要です（ ps7_system_prj.xmlとして使用するため）。このファイルには、 MIO ピンの事実上の使用や DDR ピンの電気的パラメータなど、 processorの設定が含まれています。推奨される方法は、少なくとも開始点として、参照ファイルを採用することです。

- XML ファイルを参照として採用する場合は、参照ファイルの内容に関係なく、 FPGA CLK1 （FCLK_CLK1）を100 MHzに設定する必要があります。
- 手動で変更を行う場合は、 processor coreの MIO 割り当てに注意を払う必要があります。 ARM core には54個の I/O pins があり、固定配置でチップ上の物理ピンにルーティングされます。 ARM core は、プロジェクトの block design で構成され、これらのピン（ USB インターフェイス、 Ethernet など）に特定の役割を割り当てます。これらの役割は、これらのピンがボード上で配線されているものと一致する必要があります。
- processorの構成（つまり XML file）に変更が加えられた場合は、新しい XML ファイルから派生した FSBL (First Stage Boot Loader) と U-boot binaryに基づいて、 boot.bin を再構築する必要があります。 Vivadoの block design ツールで行われた変更は、 FSBLの一部である初期化ルーチンを通じて有効になります。このルーチンは、 Vivadoで行われた設定を反映する値を使用して、 ARM processorの registers に書き込み、 SDKにエクスポートします。 Vivado プロジェクトの processor のパラメーターは正確でない可能性があるため、バンドルで使用可能な XPS プロジェクトに基づいて FSBL を生成する必要があることに注意してください。 U-bootのソースを設定するには、 /usr/src/xillinux/uboot-patches/の README ファイルを参照してください。
- あるいは、 “poke” の機能により、 registersの設定の変更を特定することで、 boot.bin の再構築を回避できる場合もあります。 5.6項を参照してください。
- 新しい設定を反映するために、 devicetree.dtbで変更を加える必要がある場合もあります。既存の DTB （ DTS 形式）のソースは、 Linux kernel のソースにあります（段落 6.2を参照）。
- VGA/DVI 出力（該当する場合）は、目的のボードに一致させる必要があります。これは、 src/ サブディレクトリの xillybus.v ファイルを編集することによって行われます。 “system” モジュールから到着する信号は8ビット幅であり、4ビットへの切り捨ては xillybus.vで行われることに注意してください。したがって、これらの信号を VGA/DVI用のエンコーダチップに接続するのはかなり簡単です。

5.3 システム内の clocks の周波数を変更する

ARM processorの core は、一般に FCLK_CLKnと呼ばれる logic fabricで使用する 4 つの clocks を提供します。 U-boot がロードされる前に、周波数が FSBL (First Stage Boot Loader)によって設定されることに注意することが重要です。

したがって、 clocksの周波数は Vivadoで設定されていますが、これらの周波数は、 timing constraints の伝搬と、 SDK上の compiled である bare-metal アプリケーションによる初期化にのみ有効です。

ハードウェアアプリケーションが異なる周波数を必要とする場合は、次の一連のアクションが推奨されます。

- Vivadoの clock の周波数を更新します。
- ネットリストを再構築します（これは、 .ncf ファイルの timing constraints を更新するために必要です）
- プロジェクトを SDKにエクスポートし、これに基づいて FSBL application project を作成します。
- Vivadoのレポートから、目的の設定に必要な registersの設定を確認してください。
- 5.6項で説明されているように、 “poke” 機能を使用して、必要に応じて調整します。

各ステップの実行方法の詳細については、 Xilinxのガイドを参照してください（最後のステップを除く）。

5.4 PL logicの GPIO I/O ピンを引き継ぐ

5.4.1 Z-Turn Lite

Z-Turn Lite ボード自体は、ラボの目的で I/O ピンにアクセスする便利な方法を提供していませんが、 Z-Turn Lite IO Cape board に接続すると、 HDMI インターフェイスに加えて、標準コネクタを介して 68 I/O pins とプッシュボタンが露出します。

これらの68ピンはすべて、標準のフラットリボンケーブルを接続できる2つの40ピンコネクタ J3 および J8に配線されています。 IO Cape board には、 J3 および J8とピンを共有するいくつかの追加コネクタがあります。追加のコネクタのピンはすべて J3 および J8で使用できるため、これらのピン用の Xillydemoの top-level moduleのポートは、 J3 および J8という名前のベクトルであり、 pin placement constraints はそれらを対応するコネクタにルーティングします。

J3 と J8 の両方の Verilog / VHDL のポートのベクトルは、コネクタのピン番号から3を引いたものに対応します。 Verilog / VHDL の信号 J3[0] は、物理ピン J3/3に送られます。 J3[1] は J3/4 などに接続され、 J3[33] は J3/36に接続されます。同じことが J8にも当てはまります。

簡単にするために、 J8 に属するすべてのピンは xillydemo.v および xillydemo.vhd で processorの GPIO ピンに接続され、 processorで実行されているソフトウェアから直接制御できます。 J3 のすべてのピンは xillydemo.v および xillydemo.vhdによってローに駆動され、関連する xillydemo モジュールファイルを変更することにより、アプリケーション logic で簡単に利用できます。

GPIO へのピンと application logic によって駆動されるピンのこの分割も、 xillydemo モジュールの配線を変更することで簡単に変更できます。 GPIO として使用されるピンの数を変更する場合は、それに応じて xillybus モジュールの gpio_width インスタンス化パラメーター（汎用）を変更する必要があります。現在35であり、 J8 コネクタに接続される34の I/O ピンと、 Cape Boardの押しボタン用の1つの GPIO を占めています。

また、前述のように、ボード上には J3 および J8とピンを共有する追加のコネクタがあり、これらは引き続き使用できます。ただし、これには、 Cape Boardの回路図のどこにどのピンが配置されているかを調べる必要があります。

このピン共有の副作用の1つは、代替コネクタによって I^2C として使用される一部のピンの Cape boardに pull-ups があることです： J3[19]、 J3[18]、 J8[28] 、および J8[31] （ Verilog / VHDL ベクトル信号表記を使用）。これらはボード上に抵抗を備えた pull-ups であるため、 J3 および J8 コネクタでこれらのピンを使用する場合でも有効です。

HDMI コネクタは独立しており、 J3、 J8 、または他のコネクタとピンを共有しません。

5.4.2 Zedboard および Zybo

Zedboard および Zybo ボードでは、多くの物理 I/O ピンが ARM processorの GPIO ポート（PS）に接続されているため、これらのピンを Linuxから直接制御および監視できます。ただし、これらの物理ピンを代わりに FPGA logic （つまり、 PL）に接続することが望ましい場合がよくあります。

I/O に Zynqの PL ピンを使用する手法は、他の Xilinx FPGAとまったく同じです。信号はトップレベルモジュール（xillydemo.v または xillydemo.vhd）で入力、出力、または inoutsとして公開されます。これらの信号への物理ピンの割り当ては、xillydemo.xdcで行われます。

ピンは GPIO 信号として使用されるため、 processor から取り外され、 PL パーツに渡されます。たとえば、 XDC ファイルの次の行である my_output をピン U5に表示する場合は、

```
set_property -dict "PACKAGE_PIN U5 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[55]"]
```

を

```
set_property -dict "PACKAGE_PIN U5 IOSTANDARD LVCMOS33" [get_ports "my_output"]
```

に置き換えることができます。

ただし、この置換を行うと、 PS_GPIO[55] にピン割り当てがなくなります。 Xilinxのツールが implementation中にこのポートを自動的に配置する可能性がありますが、削除された PS_GPIO には I/O ピンを割り当てることをお勧めします。別の方法は、以下で説明するように、信号を除去することです。

したがって、これらの排除された PS_GPIO 信号には2つの解決策があります。

- 簡単な方法：デバイス上の未使用のピンを見つけて、これらのピンを削除された PS_GPIO 信号に割り当てます。これはあまりクリーンなソリューションではありませんが（GPIO ピンはボード上のすべてのものに接続されています）、 GPIOs はデフォルトで入力であるため、実質的に無害です。 GPIO が誤ってソフトウェアによって駆動されない限り、これらのピンの電気的状態は維持されます（可能性は低いです）。たとえば、 Zedboardでは、 FMC コネクタが多くの未使用のピンを供給します。
- より難しい方法： PS_GPIO ピンの数を減らす。これは、空のピンが少ない Zyboで必要になる場合があります。

以下では、2番目の解決策について説明します。たとえば、ピンを PLからの信号に置き換えるために、 PS_GPIO[55:48] が XDC ファイルから削除されたと仮定します。低い PS_GPIO インデックスのピンが必要な場合は、削除された PS_GPIO 信号が、最も高いインデックスのピンを引き継ぐ必要があり、後者は削除されることに注意してください。 PS_GPIO インデックスの特定の範囲を削除する可能性はなく、最大インデックスを減らすだけです。

XDC ファイルにピン割り当てがあるものを反映するために、 xillydemo.v/vhd では PS_GPIO の幅を小さくする必要があります。

ただし、これだけでは不十分です。この状態でプロジェクトをビルドしようとすると、これらのピンに対して critical warnings が発行されます（ high-Z および GNDで、これらが複数駆動されていると主張する可能性があります）。

これを解決するには、次の部分で vivado-essentials/system.v を編集します。

```
generate
  for (i=0; i<56; i=i+1)
    begin: gpio
      assign gpio_tri_i[i] = processing_system7_0_GPIO[i];
      assign processing_system7_0_GPIO[i] = gpio_tri_t[i] ? 1'bz :
                                            gpio_tri_o[i];
    end
endgenerate
```

インデックス範囲（つまり、 i<56 パーツ）を使用されている GPIOs の数（例では48）に減らします。

Vivadoのリビジョンによっては、信号の幅を調整する必要がある場合もあります。

block designで GPIO の幅を修正する必要がある場合： Vivadoのメインウィンドウで、左側の列の[“Open Block Design”]をクリックします。 processor block（processing_system_7_0、 ZYNQ マーキング付き）を右クリックし、 “Customize block”を選択します。左側の列で “MIO Configuration” を選択し、 “I/O Peripherals” 階層を展開して、 GPIO 階層（下部）を展開します。 EMIO GPIO (Width) パラメータは現在、 GPIO ピンの数である56になっています。必要な数（この例では48）に減らします。

5.5 7020 MicroZedでの作業

MicroZed で使用可能な boot partition kit は、デフォルトで 7010 MicroZed ボードを対象としています。ただし、 Vivado プロジェクトの作成に使用される xillydemo-vivado.tcl ファイル (つまり、選択した言語に応じて、バンドル内の verilog/xillydemo-vivado.tcl または vhdl/xillydemo-vivado.tcl) に小さな変更を加えた後、 Vivadoを使用して 7020 MicroZed を操作することは可能です。

キットを解凍した直後（および Vivadoで使用する前）に、ファイルを編集して、次のように行を変更する必要があります。

```
set thepart "xc7z010clg400-1"
```

（11行目あたり）から

```
set thepart "xc7z020clg400-1"
```

残りのビルドプロセスはまったく同じです。

5.6 hardware registers のboot 以前の操作（“poke”）

boot.bin ファイルを再構築せずに、 ARM processorのハードウェア設定にわずかな変更を加えることが望ましい場合がよくあります (段落 5.3 では、典型的な再構築シーケンスについて説明しています)。

たとえば、 processorの MIO/EMIO 構成にわずかな変更を加えると、 registersの設定にいくつかの変更が加えられます。これは、システムの設定をソフトウェアツールにエクスポートするときに生成されるレポートの違いを探すことで簡単に推測できます。

processorの hardware registers は、 TRM または ug585とも呼ばれる Xilinxの Zynq-7000 AP SoC Technical Reference Manualに記載されています。

registersを操作するために、エントリが kernelの device tree に追加されます（通常、 Linux kernel ソースで提供されているそれぞれの DTS ファイルを編集します。段落 6.2を参照してください）。

次の例のようなエントリは、 device treeの階層の任意の場所に追加されます（できれば “chosen” エントリの後に）。

```
poke {
        compatible = "xillybus,poke-1.0";
        sequence = < 0 0xf8002000 0
                     0 0xf800200c 0
                     0 0xf8002018 0
                     1 0xf800200c 0x20
                     0 0xf8002018 0
                     0 0xf8002018 0
                     1 0xf800200c 0x21
                     0 0xf8002018 0
                     0 0xf8002018 0
                   >;
};
```

“sequence” パーツは、 register の読み取りと書き込みの目的のシーケンスを設定するように変更する必要があります。各操作は、要素の “sequence” 配列の3つの値によって定義されます。トリプレットの数（したがって操作）に制限はありません。tabs と行ごとに3つの値を使用する、上記の “sequence” エントリのフォーマットには、構文上の重要性はありません。重要なのは、各トリプレットが次のように操作を表すことです。

- 最初の要素：読み取りまたは書き込み。値0は読み取りを意味し、それ以外の

場合は書き込みを意味します。

- 2番目の要素：アドレス。 32ビットで整列されている必要があります（アドレスの2 LSBs はゼロである必要があります）。
- 3番目の要素：書き込む値。読み取り操作では無視されます。

操作は、 device tree エントリにリストされている順序で実行され、各操作の間に予測できない遅延が発生します。

上記の例では、実用的な意味はありませんが、 processorの ttc2 (Triple Timer Counter 2) の registers が操作されます。最初の3つの操作は、デモンストレーションのために registers を読み取ります。次に、カウンターが短時間有効になり、カウンター値が2回読み取られて変化していることを示してから、カウンターが無効になります。この後、カウンター値が再度2回読み取られ、停止したことが示されます。

これらの操作の結果は、 serial console (UART)で使用可能な kernelの message log、および/または shell promptの dmesg コマンドで確認できます。

```
[    0.000000] poke read addr=f8002000: value=00000000
[    0.000000] poke read addr=f800200c: value=00000021
[    0.000000] poke read addr=f8002018: value=00000000
[    0.000000] poke write addr=f800200c: value=00000020
[    0.000000] poke read addr=f8002018: value=00000009
[    0.000000] poke read addr=f8002018: value=00004f68
[    0.000000] poke write addr=f800200c: value=00000021
[    0.000000] poke read addr=f8002018: value=000013ec
[    0.000000] poke read addr=f8002018: value=000013ec
```

もちろん、 0xf8002018 から読み取られる値は、実行中のカウンターから読み取られるため、異なります。

register の変更は、 device driver がロードされる前に、 kernelの boot プロセスの早い段階で行われます。ただし、 Linux が boot プロセスを開始する前に、 U-boot が ARM processorのハードウェア周辺機器のいくつかをセットアップするため、それらはすでにアクティブになっていることに注意してください。また、 ARM processorの基本機能 (clocks や interruptsなど) に関連する registers を変更すると、 processor 自体の適切な機能が損なわれる可能性があることにも注意してください。これは、 “poke” の実行時に kernel によって interrupts が無効にされていても発生する可能性があります。

kernelのメモリ管理またはハードウェア自体のいずれかによって許可されていないアドレスにアクセスしようとすると、 kernel Oops、 kernel panic が発生し、場合

によっては完全にフリーズします。後者の2つは、 boot 障害を引き起こします。 “imprecise external abort (0x406)” を理由とする kernel panic は、ハードウェア的に不正なアドレスにアクセスしようとしたことが原因である可能性があります。

さらに、初期の kernel console メッセージは、生成された段階で内部 memory buffer に保存され、後の段階（ serial port のセットアップ時）でのみ console に書き込まれるため、早期のフリーズによって出力がなくなる可能性があります。 console にまったく- U-bootの “done, booting the kernel” メッセージの後に何も表示されません。

したがって、 kernel messages が consoleに表示されない場合、必ずしも kernel が起動しなかったことを意味するわけではありません。これは、メモリに格納されていた kernel messagesが consoleに書き込まれる前のフリーズの結果である可能性があります。その理由は、 registerの違法な変更である可能性があります。

“poke” 機能は、 Xillinux-2.0 の kernel に特にパッチを適用することによって追加されたものであり、メインライン Linux kernelsの一部ではありません。

6

Linux ノート

6.1 全般的

このセクションには、 Xillinuxに固有の Linux関連のトピックが含まれています。

Xillybus および Linux のより広いビューは、次のドキュメントに記載されています。

- Getting started with Xillybus on a Linux host
- Xillybus host application programming guide for Linux

6.2 Linux kernelのCompilation

そのサイズのため、完全な Linux kernel は Xillinux ディストリビューションには含まれていませんが、 Github からダウンロードできます。

```
$ git clone https://github.com/xillybus/xillinux-kernel.git
```

Xillinux-2.0で使用される kernel の正確な複製については、 “xillinux-2.0” タグを確認してください。目的の cross compiler 上の kernel ビルド環境に次の形式で通知します。

```
$ export CROSS_COMPILE=/path/to/crosscompiler/arm-xilinx-linux-gnueabi-
```

次に、 Xillinux-2.0に付属する kernel の compilation に使用される .config ファイルを取得します。

```
$ make ARCH=arm xillinux_defconfig
```

次に、 kernelの compilation を実行します。

```
$ make ARCH=arm -j 8 uImage modules LOADADDR=0x8000
```

Device Tree Blobsをビルドするには、上記のように cross compiler および .config ファイルを設定した後、次のコマンドを使用できます。

```
$ make ARCH=arm dtbs
[ ... ]
  DTC     arch/arm/boot/dts/xillinux-microzed.dtb
  DTC     arch/arm/boot/dts/xillinux-zedboard.dtb
  DTC     arch/arm/boot/dts/xillinux-zybo.dtb
  DTC     arch/arm/boot/dts/xillinux-zturn-lite.dtb
```

DTB ファイルは、コマンドの応答で示されているように、 arch/arm/boot/dts/にあります。

6.3 kernel modulesのCompilation

Xillinux ディストリビューションには、実行中の kernelの compilation headersが付属しています。これは、 kernel 自体の compilation を実行するには十分ではありませんが、 kernel modules の compilation をプラットフォーム上で直接実行できます。

ツリー外の kernel module の compilation の標準的な方法は、特定のモジュールの compilation を実行するように指示する environment variableをセットアップした後、 kernel自体のビルド環境を呼び出す Makefile を使用することです。

単一のソースファイル example.cで構成される kernel module のネイティブ compilation （ Zynq processor 自体によって実行される）の最小 Makefile は、次のとおりです。

```
ifneq ($(KERNELRELEASE),)
obj-m := example.o
else
TARGET := $(shell uname -r)
PWD := $(shell pwd)
KDIR := /lib/modules/$(TARGET)/build

default:
        @echo $(TARGET) > module.target
```

```
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
endif
```

モジュールの名前 “example”は、 “obj-m” 行でのみ言及されていることに注意してください。これは、 Makefile ごとに異なる唯一のものです。

この Makefile を使用すると、通常、次のような compilation セッションが発生します（ボード自体で実行されます。これは cross compilationではありません）。

```
# make
make -C /lib/modules/4.4.30-xillinux-2.0/build SUBDIRS=/root/example modules
make[1]: Entering directory '/usr/src/linux-headers-4.4.30-xillinux-2.0'
  CC [M]  /root/example/example.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/example/example.mod.o
  LD [M]  /root/example/example.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.30-xillinux-2.0'
```

6.4 サウンドサポート

6.4.1 全般的

Zedboard および Zybo ボードは、それぞれ Analog Devicesの ADAU1761 および SSM2603 チップセットによる録音と再生をサポートします。これらは、 Zynq デバイスの logic fabric (PL) ピンにのみ接続されます。

明らかな機能を除いて、サウンドサポートパッケージは、 Xillybus IP core を使用してデータを転送したり、 SMBus / I^2Cを介してチップをプログラムしたりする方法も示しています。

Xillinux は、専用の Xillybus streams を、現在の Linux のサウンド用の最も一般的なツールキットである Pulseaudioとインターフェースすることにより、ネイティブにサウンドをサポートします。その結果、サウンドカードを必要とする事実上すべてのアプリケーションは、システムのデフォルトの入力および出力としてボードのサウンドチップを適切に使用します。

Pulseaudio daemon をオフにして、 Xillybus streams （/dev/xillybus_audio）を直接操作することもできます。これは、他のアプリケーションのネイティブ機能を失うという犠牲を払って、 device fileを開くだけのより単純なプログラミングインターフェイスを提供します。

Linux サウンドカードの一般的なアプローチとは異なり、 Xillybus host driver はデータ転送を処理するため、サウンドインターフェイス専用の kernel driver （ ALSAなど）はありません。 Pulseaudio には、 “padsp” ユーティリティを使用してこのインターフェイスを偽造する機能があるため、 /dev/dspでの動作を期待しているプログラムにとっても、これは重要ではありません。

6.4.2 使用法の詳細

デフォルトでは、サウンドはヘッドホンジャック（黒）で再生されます。 Zedboardでは、同じ出力が Line Out （緑）にも送られます。録音にはマイク入力（ピンク）のみを使用しますが、次に説明するように変更することができます。

6.4.3 関連する boot scripts

Zedboard と Zyboでのサウンドのセットアップに関連する2つのタスクがあります。オーディオチップのプログラミングと Pulseaudio daemonの起動です。

最初のタスクでは、2つの systemd unit files が /etc/systemd/system/に配置されています。

- xillinux_sound.service：起動時に /usr/local/bin/xillinux-sound を起動します。
- xillinux_sound.path： /dev/xillybus_smb が表示されるのを待つように systemd に指示し、その場合は、前述の service を起動します。これは udevに基づくのではなく、 /devを監視している inotifyに基づいていることに注意してください。

/usr/local/bin/xillinux-sound は、実行するボードを識別し、必要に応じて script /usr/local/bin/zybo_sound_setup.pl または /usr/local/bin/zedboard_sound_setup.plを実行します。

Pulseaudio daemon は、 /etc/systemd/user/に存在するユーザーサービスユニットファイル xillinux_pulseaudio.serviceにより、ユーザーがシステム上で最初のログインセッションを作成したときに開始されます。

Xillinux は、シリアル console および画面 consoleで root ユーザーに自動ログインするため、システムの boot が完了した直後に Pulseaudio が開始されます。

Pulseaudio を root として実行することは、マルチユーザーコンピューターでは推奨されませんが、 Xillinux はデフォルトユーザーとして root で実行されるため、これは最も問題の少ないオプションです。

/dev/xillybus_audio に直接アクセスする必要がある場合は、サービスを無効にする必要があります。

```
# systemctl --global disable xillinux_pulseaudio
Removed symlink /etc/systemd/user/default.target.wants/
xillinux_pulseaudio.service.
```

zybo_sound_setup.pl と zedboard_sound_setup.pl は Perl scriptsであり、オーディオチップの registers を適切に動作するようにセットアップします。 Perlに精通していないプログラマーにとっても、これらはかなり簡単です。 script は、/dev/xillybus_smbus device file を使用して、チップの I^2C busでトランザクションを開始します。

zedboard_sound_setup.pl を編集して、オーディオチップの異なるセットアップを実現できます。特に、 Line In 入力が録音に必要なソースである場合は、

```
write_i2c(0x400a, 0x0b, 0x08);
write_i2c(0x400c, 0x0b, 0x08);
```

行を次のように置き換える必要があります。

```
write_i2c(0x400a, 0x01, 0x05);
write_i2c(0x400c, 0x01, 0x05);
```

同様に、 zybo_sound_setup.pl を編集して、

```
write_i2c(0x04, 0x14);
```

を

```
write_i2c(0x04, 0x10);
```

に置き換え、 Line In を記録に使用することができます。

6.4.4 /dev/xillybus_audio に直接アクセスする

/dev/xillybus_audio は、再生用に直接書き込むことも、録音用に読み取ることもできます。 sample format はオーディオサンプルあたり32ビットで、 little Endian 形式の2つの16ビット signed integers に分割されています。最も重要な単語は左チャネルに対応します。

サンプリングレートは 48000 Hzに固定されています。

このサンプリングレートの Windows WAV ファイルは、 /dev/xillybus_audio に直接書き込まれた場合に正しく再生される可能性があります（ header は約 1 msでも再生されます）。

```
# cat song.wav > /dev/xillybus_audio
```

応答が

```
-bash: /dev/xillybus_audio: Device or resource busy
```

の場合、別の process が device file を書き込み用に開いている可能性があります（ Pulseaudio daemonの可能性があります）。 device file を開いて、あるプロセスによる読み取りと別のプロセスによる書き込みを行うことは問題ありません。

6.4.5 Pulseaudio の詳細

Pulseaudio は、いくつかの専用 Pulseaudio モジュール、 module-file-sink および module-file-sourceを介して /dev/xillybus_audio device file と対話します。それらのソースは、 Xillinuxの file system の /usr/src/xillinux/pulseaudio/にあります。

これらのモジュールは、 UNIX pipes をデータ sinks およびソース（module-pipe-sink および module-pipe-source）として使用するための標準 Pulseaudio モジュールのわずかな変更です。

モジュールは、 /etc/pulse/default.paの次の2行により、 Pulseaudio の起動時に自動的にロードされます。

```
load-module module-file-sink file=/dev/xillybus_audio rate=48000
load-module module-file-source file=/dev/xillybus_audio rate=48000
```

これらのモジュールは、他に選択肢がないため、システムのサウンドインターフェイスとして自動的に選択されます。

6.5 OLED ユーティリティ（Zedboard のみ）

デフォルトでは、 Xillinux は boot中にアクティビティメーターユーティリティを開始し、およそ CPU の使用率と SD flash diskの I/O レートを表示します。

CPU のパーセンテージは、 /proc/statに基づいており、アイドル状態になっていない時間はすべて CPU の使用時間と見なされます。

SDIO トラフィックの推定は、割り込みがそれぞれの driverに送信される速度に基づいて行われます。このリソースを完全に活用するための既知の数値はありません。むしろ、ユーティリティは、以前の測定によれば、最大と思われる割り込みレートに対して 100% を表示します。

ボードの OLEDにグラフィック出力を表示するために、 bitmap は Digilentの driverによって作成された /dev/zed_oledに送信されます。この driver は、 SPI データを bit-banging メカニズムを備えた OLED デバイスに送信し、 clock とソフトウェア内のデータを切り替えます。したがって、この方法で1秒間に数回512バイトを送信する CPU の消費量はわずかですが、システム全体のパフォーマンスへの影響は最小限です。

このユーティリティのパラメータを変更するには、 /usr/local/bin/start_zedboard_oledを編集します。これは、次のコマンドに要約されます。

```
/usr/local/bin/zedboard_oled /proc/irq/$irqnum/spurious 4 800
```

アプリケーション zedboard_oledは、次の3つの引数を取ります。

- SDIO関連の割り込みを監視するための /proc ファイル。 $irqnum は、 mmc0 デバイスの IRQ 番号です。
- OLED ディスプレイが更新される速度（1秒あたりの回数）。
- SDIO 割り込みレートの 100% と見なされるもの（1秒あたりの割り込み数）。現在の数字は試行錯誤によって発見されました。

boot中にこのユーティリティが起動されないようにするには：

```
# systemctl disable zedboard_oled.path
Removed symlink /etc/systemd/system/paths.target.wants/zedboard_oled.path.
```

6.6 その他の注意事項

- kernel 3.12以降 Linux GPIO driver に変更がありますが、 Xillinux-1.3 と同じ GPIO driver が Xillinux-2.0で使用され、 Xillinux-1.3の動作と番号付けが保持されます。

  新しい GPIO driverを使用するには、 device tree エントリを compatible = “xlnx,ps7-gpio-1.00.a”に変更して、 “xlnx,zynq-gpio-1.0”と表示します。

- driver はクアッドビットインターフェイスをサポートしていますが、 Quad SPI flash は1ビット幅の busを備えた Linux kernel によってアクセスされます。関連する device tree エントリは、クアッドビットインターフェイスを有効にするために変更できます。これは、同じ device tree を kernel 3.12（つまり、 Xillinux-1.3）で使用できるようにするためのデフォルト設定ではありません。

7

トラブルシューティング

7.1 implementation中のエラー

Xilinxのツールのリリース間のわずかな違いにより、 bitfileを作成するための implementation の実行に失敗することがあります。

問題がすぐに解決されない場合は、 Xillybusのフォーラムで支援を求めてください。

http://forum.xillybus.com

失敗したプロセスの出力ログ、特にツールによって報告された最初のエラーの前後に添付してください。また、 designでカスタム変更が行われた場合は、これらの変更について詳しく説明してください。また、使用された ISE / Vivado ツールのバージョンを明記してください。

このエラーが VHDL用の implementation 上の Vivado によって発行された場合：

```
ERROR: [Place 30-58] IO placement is infeasible. Number of unplaced terminals (3) is greater than number of available sites (2).
The following Groups of I/O terminals have not sufficient capacity:
Bank: 35:
 The following table lists all user constrained IO terminals
 Please analyze any user constraints (PACKAGE_PIN, LOC, IOSTANDARD ) which may cause a feasible placement to be impossible.
 The following table uses the following notations:
 c1 - is IOStandard compatible with bank? 1 - compatible, 0 is not
 c2 - is IO VREF compatible with INTERNAL_VREF bank? 1 - compatible, 0 is not
 c3 - is IO with DriveStrength compatible with bank? 1 - compatible, 0 is not
+--------+------------+----+----+----+-----------------------------------------------------+
| BankId | IOStandard | c1 | c2 | c3 | Terminal Name                                       |
+--------+------------+----+----+----+-----------------------------------------------------+
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_CLK                                            |
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_PORB                                           |
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_SRSTB                                          |
+--------+------------+----+----+----+-----------------------------------------------------+
```

3.3項に記載されているように xillydemo.vhd を編集してください。

同じ問題に起因する別の考えられるエラー：

```
ERROR: [Drc 23-20] Rule violation (NSTD-1) Unspecified I/O Standard
        ...
        Problem ports: PS_CLK, PS_PORB, PS_SRSTB.
ERROR: [Drc 23-20] Rule violation (RTSTAT-1) Unrouted net ...
ERROR: [Drc 23-20] Rule violation (UCIO-1) Unconstrained Logical Port ...
```

7.2 USB キーボードとマウスの問題

ほとんどすべての USB キーボードとマウスは、互換性のある動作の標準仕様を満たしているため、認識されないデバイスで問題が発生する可能性はほとんどありません。何か問題が発生したかどうかを最初に確認することは次のとおりです。

- Zedboard のみ：正しい USB プラグを使用していますか？これは、電源スイッチから離れた、“USB OTG”とマークされたものである必要があります。
- Zedboard のみ：デバイスへの 5V の供給はありますか？ JP2 ジャンパーはインストールされていますか？ Zedboard の電源がオンの状態で、光学式 USB マウスを接続し、 LED がオンになることを確認します。
- USB hub を使用する場合は、キーボードまたはマウスのみをボードに直接接続してみてください。

役立つ情報は、一般的なシステムログファイル /var/log/syslogに含まれている場合があります。 “less /var/log/syslog” でコンテンツを表示すると、役立つ場合があります。さらに良いことに、 “tail -f /var/log/syslog” と入力すると、新しいメッセージが到着すると console にダンプされます。 USB bus のイベントは、検出された内容とイベントの処理方法に関する詳細な説明を含め、常にこのログに記録されるため、これは特に便利です。

shell prompt は USB UARTからもアクセスできるため、キーボードの接続に失敗した場合でも、シリアル端末でログを表示できることに注意してください。

7.3 file system mountの問題

経験によれば、適切な (Micro)SD カードが使用され、ボードの電源を切る前にシステムが適切にシャットダウンされた場合、 (Micro)SD カードのデータにまったく問題はありません。

ext4 file system は次の mountの journal で修復されるため、 root file system の unmounting を使用せずにボードの電源をオフにしても、 file system 自体に永続的

な不整合が発生する可能性はほとんどありません。ただし、電源がオフになったときに書き込み用に開かれたファイルには、誤った内容が残ったり、完全に削除されたりする可能性があるため、オペレーティングシステムの機能に累積的な損傷があります。これは、突然電源がオフになったすべてのコンピューターに当てはまります。

root file system が mount の実行に失敗した場合（ boot中に kernel panic が発生した場合）、または mount を read-onlyとしてのみ実行した場合、最も可能性の高い原因は低品質の (Micro)SD カードです。このようなストレージがしばらくの間適切に機能し、その後ランダムなエラーメッセージが表示されるのは非常に一般的です。/var/log/syslog に次のようなメッセージが含まれている場合は、 (Micro)SD カードが原因である可能性があります。

```
EXT4-fs (mmcblk0p2): warning: mounting fs with errors, running ec2fsck
   is recommended
```

これらの問題を回避するには、 Sandisk デバイスを主張してください。

7.4 “startx” が失敗します（Graphical desktop は起動しません）

直接関係はありませんが、 (Micro)SD カードが Sandiskで製造されていない場合、この問題は非常に頻繁に報告されます。グラフィカルソフトウェアは、起動時にカードから大量のデータを読み取るため、読み取りエラーを生成する (Micro)SD カードの顕著な被害者になる可能性があります。

明らかな解決策は、 Sandisk (Micro)SD カードを使用することです。

7.5 X desktopでスクリーンセーバーを実行した後の黒い画面

/root ディレクトリがクリアされた場合、新しいユーザーがデスクトップを使用している場合、または省電力設定が変更された場合、デスクトップはスクリーンセーバーモードから回復せず、再開しようとすると黒い画面が残ることがあります。

修正： XFCE desktopで、 Power Managerに移動し、次の設定を行います。

- Display > “Put to sleep after” を “Never”に設定します
- Display > “Put to sleep after” を “Never”に設定します
- Display > “Switch off after” を “Never”に設定します。
- Security > “Automatically lock the session” を “Never”に設定します。

これらの設定は root ユーザー用にすでに設定されているため、この問題は新しい Xillinux ディストリビューションでは発生しないはずです。