(機械で日本語に翻訳)

Getting started with Xillinux for Cyclone V SoC (SoCKit)

Xillybus Ltd.

www.xillybus.com

Version 2.0

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 This document may also be slightly outdated in relation to the original.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

重要:

公共の関心が比較的低いため、 *SoCKit* 用の *Xillinux* は段階的に廃止されています。既知の問題はなく、すぐに使用できますが、その実装は、 *FPGA* バンドルを構築するためのかなり古い *Quartus II 13.0sp1* に限定されています。これは、今後変更される予定はありません。 *Cyclone V SoC*、特に *SoCKit* の全体的なサポートは制限されています。

1.1 Xillinux ディストリビューション

Xillinux は、 SoCKit ボード用の完全でグラフィカルな Ubuntu 12.04 LTS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。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。

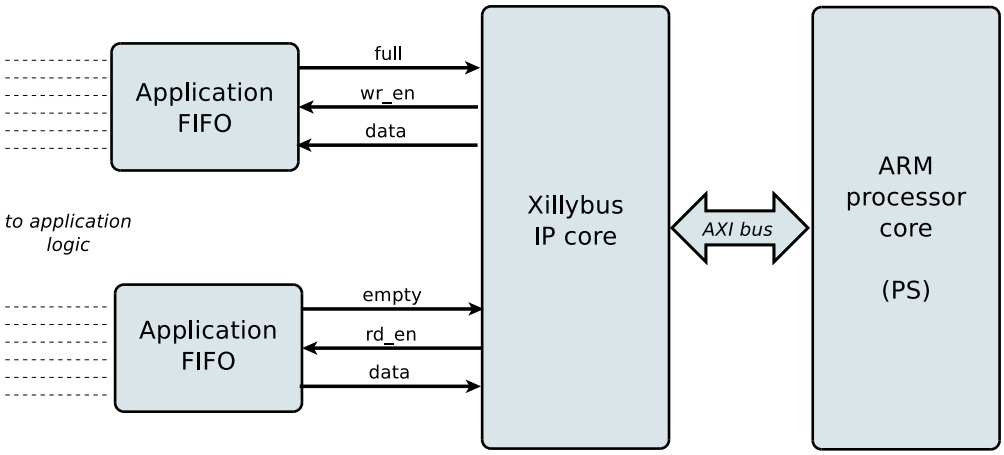
ディストリビューションは、従来のキーボード、マウス、およびモニター設定用に編成されています。また、 USB UART ポートからのコマンドライン制御も可能ですが、この機能は主に問題を解決するために利用できます。

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 ソースおよび data sinks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。

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

Sockit Linux ディストリビューション (Xillinux) の Xillybus は、現在 SoCKit ボードのみをサポートしています。

他のボードの所有者は、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、重要な変更が必要になる場合があります。これについては、セクション 5.2で詳しく説明します。

次の機器も必要です。

- アナログ VGA 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、ほぼすべての PC モニター)。
- モニター用のアナログ VGA ケーブル
- USB キーボード
- USB マウス
- Linux 3.8.0 によって認識される USB ハブ (キーボードとマウスが単一の USB プラグに結合されていない場合)
- SoCKit ボード カードへの USB ケーブル、タイプ A レセプタクル (メス) から USB マイクロ B プラグ。 SoCKit ボードの購入時には、このケーブルは**含まれていません**。
- GB が 4 つ以上搭載された信頼性の高い microSD カード。 Sandisk製のカードが最も望ましい。

  Xillinuxで使用すると問題が報告されているため、他のブランドはお勧めしません。

- イメージと boot file をカードに書き込むための microSD カードと PC間の USB アダプター。 PC コンピュータに SD カード用の組み込みスロットがある場合、これは不要な場合があります。

  SD と microSD の違いは物理的なフォーム ファクターにすぎないため、MicroSD から USB アダプターの代わりに、 SD から microSD アダプターと組み合わせて、 SD カードから USB アダプターを使用できることに注意してください。

USB ハブが不要になり、誤って USB ケーブルを引っ張ってボード上の USB ポートが物理的に損傷する可能性を防ぐため、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。

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: http://xillybus.com/xillinux/からダウンロードできます。

ディストリビューションは 2 つの部分で構成されています: 起動時に Linux によって認識される file system で構成される microSD カードの raw image と、 first-stage boot imageを生成するための Intel FPGA ツールを使用して実装するための一連のファイルです。これについての詳細はセクション 3です。

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

Xillinuxのビルドには Quartus II 13.0sp1 (Web または Subscription Edition) のみを使用できます。他のリビジョンでバンドルをビルドしようとすると、エラーで失敗します。 ARM processor は Intel FPGA デバイスにとって新しいものであるため、サポートするソフトウェア ツールは急速に変化しているため、信頼できる結果を保証する唯一の方法は、 Xillinux が適切にテストされたのとまったく同じリビジョンを使用することです。

Cyclone V SoC の Xillinux は段階的に廃止されているため、残念ながら将来的にこの制限を解除する予定はありません。

特に Windows マシンでは、 Quartus II の 64 ビット バージョンが推奨されます。ツールは、32 ビット Quartus IIでは失敗する可能性がある RAMの 2GB よりも多くを割り当てるためです。 32 ビット Windows XP が使用されている場合、 `/3gb` フラグを boot.ini に追加すると、ツールを正常に実行できます。 http://msdn.microsoft.com/en-us/library/windows/hardware/gg487508.aspxを参照してください。

32 ビット Windows 7 以降は、 `increaseuserva` パラメータで変更できます。以下を参照してください。

http://msdn.microsoft.com/en-us/library/windows/hardware/ff542202(v=vs.85).aspx.

Xillinuxのビルドに Soc EDS パッケージは**必要ありません**。 user space programs および/または kernel modules の compilation は、ボードの processor で直接実行できます。

このソフトウェアは、 Intel FPGAの Web サイト (https://www.intel.com) から直接ダウンロードできます。

2.4 FPGA designの使用経験

SoCKit ボードを使用する場合、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。別のボードで作業するには、 Intel FPGAのツールの使用に関するある程度の知識と、場合によっては Linux kernelに関連するいくつかの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。

microSD カードから Xillinux ディストリビューションの boot を実行するには、次の3 つのコンポーネントが必要です。

- U-boot ローダーで構成される、特別な partitionに常駐する初期 boot image 環境
- FPGA 部分の構成 bitstream 、 Linux kernel バイナリ、およびその device treeを含むファイルを含む小さな FAT ファイルシステム。
- Linuxに搭載された root file system 。

FPGAの bitstreamを除くこれらすべては、 Xillinuxの microSD イメージに既に含まれています。

このセクションでは、 microSD を準備するためのさまざまな操作について、順を追って詳しく説明します。ほとんどの時間は、 FPGA bitstreamの準備に費やされます。

この手順では、 SoCKit ボードが使用されていることを前提としています。これは次の手順で構成されており、以下に概説する順序で実行する必要があります。

- boot image キットの解凍
- processorのラッパーと bus インフラストラクチャの生成
- メインの FPGA プロジェクトを実装し、 bitstream を正しい形式に変換します。
- 生の Xillinux イメージを microSD カードに書き込む
- bitstream ファイルを microSD カードにコピーする

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 image キットの解凍

以前にダウンロードした xillinux-eval-sockit-*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は空白を含めないでください。特に、パスに *"Documents and Settings"*が含まれているため、デスクトップは不適切です。

バンドルは次のディレクトリ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
- core – Xillybus IP coresのプリコンパイルされたバイナリ
- soc_system – ARM processor ラッパーと bus インフラストラクチャ
- instantiation templates – Verilog および VHDLに instantiation templates が含まれています

'verilog' および 'vhdl' ディレクトリには、 SoCKit ボード用の QSF ファイルも含まれていることに注意してください。別のボードを使用する場合は、このファイルを編集する必要があります。

また、 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。

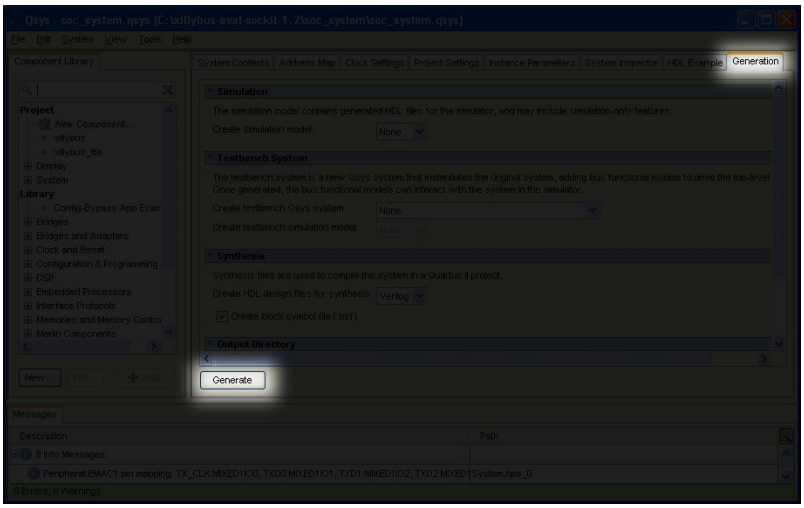
Xillybus IP core とのインターフェースは、それぞれの 'src' サブディレクトリの xillydemo.v または xillydemo.vhd ファイルで行われます。これは、独自のデータソースと sinksで Xillybus を試すために編集するファイルです。

3.3 processorのラッパーの生成

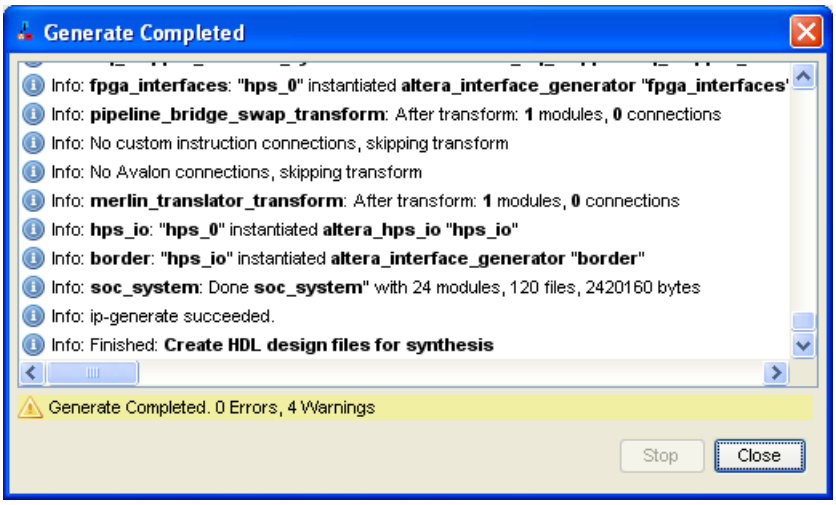
boot image キットの 'verilog' または 'vhdl' サブディレクトリにある「xillydemo.qpf」ファイルをダブルクリックして、 Quartus IIを開きます。

Quartus内で、 "File > Open..." を選択して Qsys プロジェクトを開き、1 つ上のディレクトリに移動し、 soc_system ディレクトリに入り、 soc_system.qsysを選択します。 Qsys ツールが起動し、 processor ラッパー プロジェクトが開きます。

ウィンドウの上部近くにある "Generation" タブを選択し、 "Generate" (下部近く) をクリックします。



このプロセスには数分ほどかかり、進行状況ウィンドウは次のようになります。



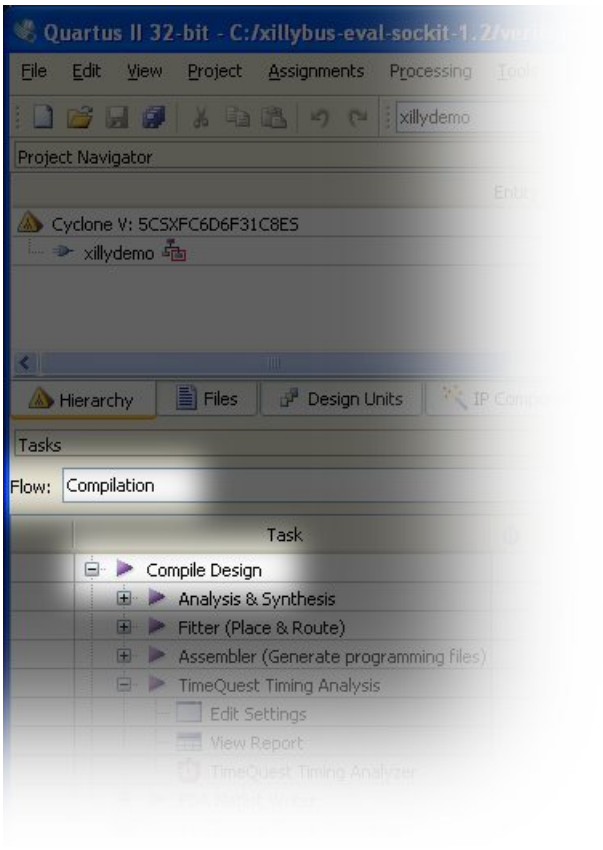
進行状況ウィンドウと Qsys ウィンドウを閉じます。今後、このプロセスを繰り返す必要はありません。

3.4 生の bitstream ファイルの生成

このステップは、 3.3の段落で説明されているように、 processorのラッパーの生成が完了した後に行われます。

好みに応じて、 'verilog' または 'vhdl' サブディレクトリにある「xillydemo.qpf」ファイルをダブルクリックします。 Quartus II が起動し、正しい設定でプロジェクトが開きます。 Qsysを使い終わったばかりであれば、すでに Quartus II を開いている可能性があります。その場合は、好みの言語が選択されていることを確認してください (気にしない場合は Verilog を選んでください)。

この画像に示すように、 “flow” が “Compilation” に設定されていることを確認し、 “Compile Design” をクリックして FPGA プログラミング ファイルを作成します。



プロセスは約 100 の警告を生成しますが、 “Full Compilation was successful”.重大な警告が生成されますが、エラーは許容されません。また、 compilationの後で、 “Timing requirements not met” (332148)という警告が**ない**ことを確認することは常に必須です。これは特に、 Verilog/VHDL ソースに独自の変更を加えた後に design の compilation を実行する場合に当てはまります。

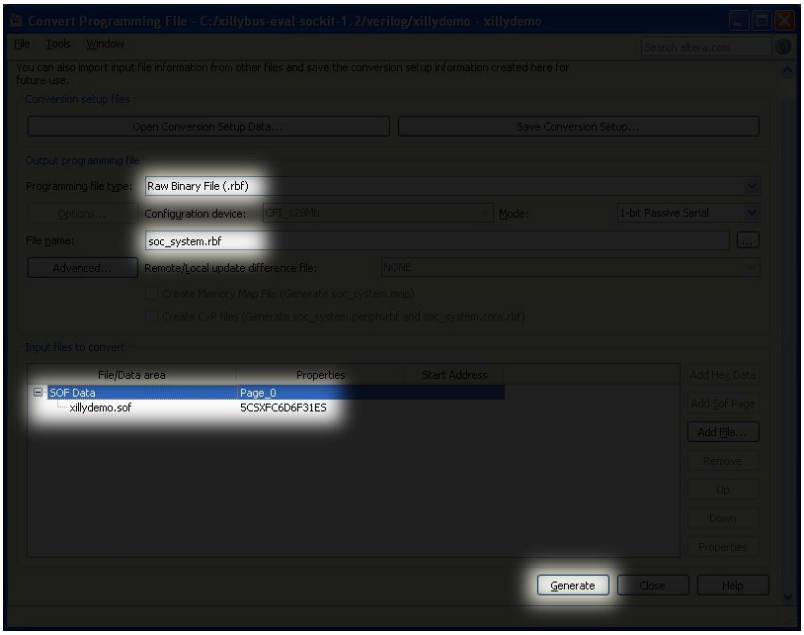
最後に、取得したプログラミング ファイルを必要な形式に変換します。 Quartus IIで File > Convert Programming Files... を選択し、プログラミング ファイル タイプとして Raw Binary File (.rbf) を選択します。すぐ下で、 “File name” を soc_system.rbfに設定します。

“Input files to convert” エリアで、 “SOF Data” をクリックしてから、右側の “Add File...” をクリックします。 xillydemo.sofを選択します。

次に、右下の “Generate” をクリックします。ファイルが正常に生成されたら、ウィンドウを閉じます。 soc_system.rbf は、段落 3.6で説明されているように、MicroSD カードにコピーする必要があります。

重要:

.rbf ファイルの生成時に圧縮を有効にしないでください *(*デフォルトのままにしてください*)*。 *soc_system.rbf* は *6 7 MBytes*である必要があります。



3.5 microSD に画像をロードする

3.5.1 全般的

このタスクの目的は、ダウンロードした microSD カード イメージ ファイルをデバイスに書き込むことです。イメージは xillinux-1.1-sockit.img.gz (または類似の) という名前のファイルとしてダウンロードされ、 microSD カードの gzip圧縮イメージです (ただし、 soc_system.rbf ファイルを追加する必要があります)。

このイメージは圧縮解除してから、 microSD 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。

イメージには、 partition table、 boot image、 raw boot partition 、および ext4 タイプの Linux root file system を配置するための部分的に実装された FAT file system が含まれています。 FAT partitions のみがほぼすべての Windows コンピュータで検出されるため、 microSD カードの容量は非常に小さいように見えます (47 MB 程度)。

フル ディスク イメージの書き込みは、通常のコンピュータ ユーザー向けの操作ではないため、 Windows コンピュータでは特別なソフトウェアが必要であり、Linuxでは特別な注意が必要です。次の段落では、どちらのオペレーティング システムでもこれを行う方法について説明します。

重要:

microSD に画像を書き込むと、画像に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。おそらく画像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。

3.5.2 画像の読み込み（Windows）

Windowsでは、 USB Image Toolなど、イメージ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 microSD 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 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。次のチュートリアルでは、 USB Image Toolの使用を想定しています。

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。左上のドロップダウン メニューで、(“Volume Mode”ではなく) “Device Mode” にいることを確認します。 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。ダウンロードしたイメージ ファイル (xillinux-1.1-sockit.img.gz) を選択します。全体のプロセスには約 4 5 分かかります。終了したら、デバイス (“safely remove hardware”) をアンマウントし、プラグを抜きます。

一部のマシンでは、 GUI が実行に失敗し、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:\         |   2014 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）

さて、デバイス番号を取得したら、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-1.1-sockit.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 microSD カードのイメージがコンピュータに保存されている場所へのパスに置き換えます。

3.5.3 画像の読み込み（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。

前述のように、正しいデバイスを microSD カードとして検出することが重要です。これは、 USB コネクタを接続することによって行うのが最適です。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します。

```
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-1.1-sockit.img.gz
```

microSD カードへの画像のコピーは簡単です。

```
# dd if=xillinux-1.1-sockit.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-1.1-sockit.img /dev/sdc
cmp: EOF on xillinux-1.1-sockit.img
```

応答に注意してください。画像ファイルで EOF に達したという事実は、他のすべてが正しく比較されたこと、および flash drive が実際に使用されたよりも多くのスペースを持ってい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、デバイスへの書き込みではなく、通常のファイル “/dev/sdc” が生成されました。

3.6 soc_system.rbf ファイルを microSD カードにコピーする

USB アダプターを取り外し、コンピューターに接続し直します。 これは、コンピュータが microSD 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。必要に応じて、最初の partition (例: /dev/sdb1) を取り付けます。ほとんどのコンピュータはこれを自動的に行います。

次に、 soc_system.rbf (段落 3.4で生成されたもの) を microSD カードの FAT file system にコピーします。 Windows システムでは、これはシステムが表示する唯一の “disk” です。 Linux システムでは、最初の (そして小さい) partitionです。いずれにせよ、正しい宛先は、既存のコンテンツ (「socfpga.dtb」と 'uImage'の 2 つのファイルのみ) によって簡単に認識されます。

完了したら、 microSD カードを適切にアンマウントし、コンピュータから取り外します。

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

SoCKit ボードが既に Xillinuxを実行している場合、ボード自体から soc_system.rbf を更新することができます。 FAT ファイルシステムを Xillinuxにマウントするには、

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

に移動し、 /mnt/sd/で file system にアクセスします。新しい soc_system.rbf 自体に問題がある場合 (圧縮されているなど)、または他の何かが不適切に行われた場合、ボードは次回 boot を実行しない可能性があることに注意してください。そのため、microSD カードにアクセスするための代替手段を常に手元に置いておく必要があります。

4

bootの実行

4.1 ジャンパーと DIP switch の設定

ボードが microSD カードから boot を実行するために、おそらくジャンパーを変更する必要はありませんが、次のページの最初の画像に示されている設定と一致していることを確認することをお勧めします。

システムの boot に関連するのは、ジャンパ J15J19 のみです。他の 2 つのジャンパーは、 LCD バックライトと HSMC インターフェイスの電圧レベルに関連しています。

DIP switches (ボードの裏側) は通常、次のページの 2 番目の画像に示すように変更する必要があります。

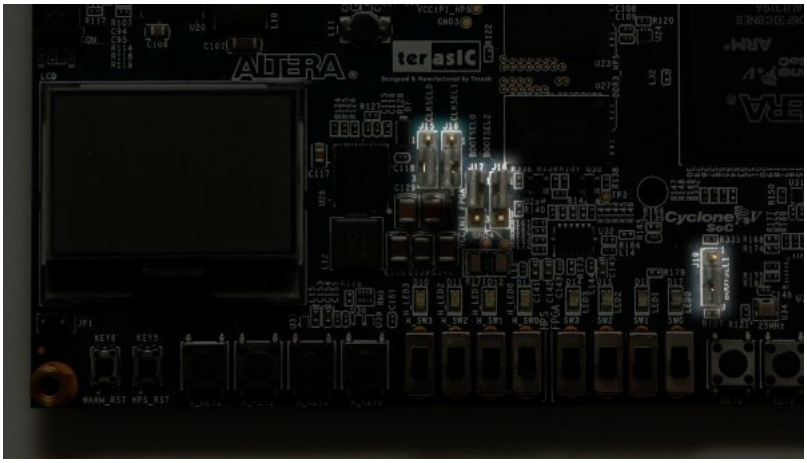
4.2 周辺機器の取り付け

重要:

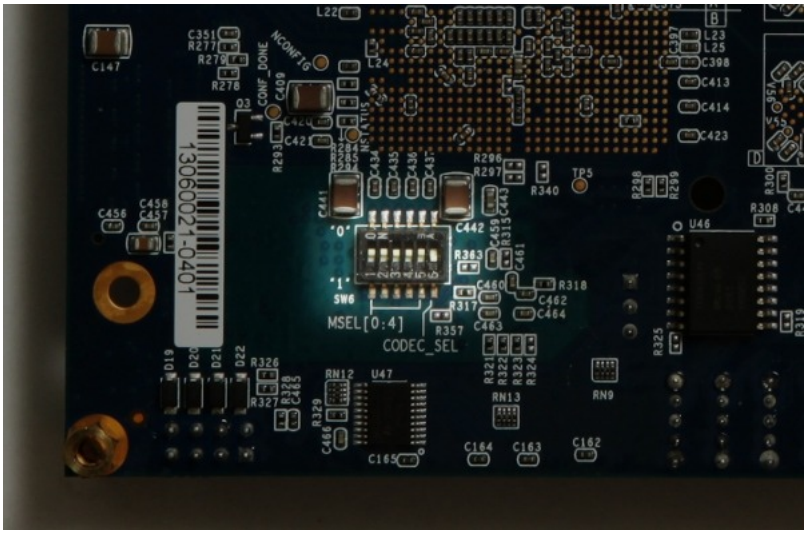
USB ポートを使用する必要がある場合 *(*例*:* マウスとキーボードを接続する*)*、*Linux* が *boot* を実行するとき *(*何も接続されていない *USB* ハブであっても*)*、いくつかの周辺機器を *OTG USB* ポートに接続する必要があります。これは、*Linux* が *processor* とそれに接続されているものの間の役割を決定するために必要です。これは、両方が *OTG* ポート上の *host* になる可能性があるためです。

次の汎用ハードウェアをボードに接続する必要があります。

- アナログ VGA コネクタへのコンピュータ モニタ。 Xillinux はアナログ VGA プラグを介して VESA準拠の 1024x768 @ 60Hz 信号を生成するため、どのコンピュータ モニタでも十分であることはほぼ確実です。



ボードの前面、ジャンパー設定が強調表示されています。



ボードの裏側、 DIP switch がハイライトされています。右端を除くすべてのスイッチが上に押し上げられます。

- USB メス ケーブル (SocKit ハードウェア キットには含まれていません) を介して、マウスとキーボードを USB OTG コネクタに接続します。これらがなくてもシステムは boot を実行し、 Linux が boot シーケンスを実行したときに何らかの周辺機器 (または単に USB ハブ) が接続されている限り、システムの実行中にキーボードとマウスを接続したり切断したりすること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。

- Ethernet ポートは、一般的なネットワーク タスクのオプションです。接続されたネットワークに DHCP サーバーがある場合、 Linux マシンはネットワークを自動的に構成します。

- UART USB ポートはオプションで PCに接続されますが、ほとんどの場合冗長です。 boot メッセージの一部はそこに送信され、 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、PC モニターまたはキーボードが見つからないか、正しく動作しない場合に役立ちます。

  このポートがコンピュータに接続されている場合、一部の terminal ソフトウェアをこのコンピュータで実行する必要があることに注意してください。そうしないと、 boot メッセージが読み取られるのを Linux が待機している間に、 boot プロセスが停止する可能性があります。このポートを接続せずに boot を実行しても問題ありません。

4.3 ボードの電源を入れる

この段落では、システムの電源を入れるときに何を期待するかについて説明します。以下の説明では、前のページの画像に示されているように、赤い電源ボタンが左上にあり、8 つの LEDs の列が下部にあるようにボードを表示しています。

重要:

Xillinuxの root file system は microSD カードに常駐し、システムが稼働している間に書き込まれます。 *Linux* システムは、ボードの電源をオフにする前に適切にシャットダウンして、通常の *PC* コンピュータと同様にシステムを安定させる必要があります。

microSD カードを SoCKit ボードに差し込み、赤いボタンを押して電源を入れます。次のシーケンスが予想されます。

- 次の LEDs はすぐにオンになります。
  - 電源ボタンの横にある緑色の LED
  - ボードの下部にある 8 つの LEDs すべて、弱い光で
  - LCD 画面のバックライト LED (このライトを無効にするジャンパーがインストールされていない場合)

  電源ボタンの少し右側でも、数回の短い点滅が予想されます。これらは UART LEDsです。コンピュータが UART USB ポートに接続されている場合は、点滅し続けます。

- 電源を入れてから 1 秒も経たないうちに、 initial bootloader が processorを初期化した兆候として、左端の 4 つの LEDs がオフになります。これが起こらない場合、考えられる原因は、 DIP スイッチまたはジャンパーが間違っているか (4.1の段落を参照)、 microSD が正しくインストールされていないか、Xillinux イメージが正しくロードされていないか、正しくロードされていないことです。

- 電源を入れてから約 14 秒後、他の 4 つの (右端の) LEDs も消灯し、右端の LED が 1 つの Hz (およそ) で点滅し始めます。 “Xillybus” スクリーンセーバーが VGA 画面に 1 秒未満表示され、空白の画面に置き換えられ、左上に 2 つの Linux ペンギンのロゴが表示されます。これが起こらない場合は、soc_system.rbf ファイルが配置され、圧縮されていないことを確認します (5-6 MBytes)。

- 電源を入れてから約 26 秒後に、 boot のテキストが VGA の画面に表示されます。

- login prompt は、電源投入後 35 秒以内に表示されます。システムは rootとして自動ログインし、グリーティング メッセージと shell promptを表示します。同様の shell prompt も USB UART リンクで提供されており、主にトラブルシューティング用です。

左端の 4 つの LEDs がオフになっても何も起こらない場合は、おそらく soc_system.rbf ファイルに問題があります。段落 4.4で説明されているように、 UARTの出力を調べると役立つ場合があります。

ボードのパワーアップを示す短いビデオ クリップは、次の場所で見ることができます。

http://youtu.be/mTDaAn4lX3I. UART USB ポートには何も接続されていないため、UARTの LEDs はクリップ内でほとんどアクティビティを示さないことに注意してください。

shell prompt で “startx” と入力して、 Gnome グラフィカル デスクトップを起動します。デスクトップの初期化には 15 30 秒かかります。

ノート：

- root userのパスワードは何も設定されていないため、必要に応じて rootとしてログインする場合、パスワードは必要ありません。
- 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。

4.4 boot中のUART 出力

boot プロセスが失敗した場合、 UARTの出力が問題のヒントを提供する場合があります。コンピューター上の terminal アプリケーションは、 57600 baud, 8 bits, 1 stop bit、 parity および flow control なし (“57600 8N1”) 用に構成する必要があります。

これは通常見られるものです:

```
U-Boot SPL 2012.10 (Dec 30 2013 - 18:03:34)
SDRAM: Initializing MMR registers
SDRAM: Calibrating PHY
SEQ.C: Preparing to start memory calibration
SEQ.C: CALIBRATION PASSED
DESIGNWARE SD/MMC: 0


U-Boot 2012.10 (Dec 30 2013 - 18:03:46)

CPU   : Altera SOCFPGA Platform
BOARD : Altera SOCFPGA Cyclone 5 Board
DRAM:  1 GiB
MMC:   DESIGNWARE SD/MMC: 0
```

```
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   mii0
Warning: failed to set MAC address
```

最後の行は数秒カウントダウンした後、自動的に 3 つのファイルを読み取ります。表示される kernel のバージョンだけでなく、バイト数も異なる場合があります。

bootloader がこれらのファイルのいずれかの読み取りエラーを報告した場合、おそらくそれが bootの実行に失敗した理由です。 “bad CRC” エラーは、 U-boot が保存された一連の環境変数を見つけられなかったことを示しているため、デフォルト値を使用していますが、これは問題ありません。

```
reading uImage

3328400 bytes read
reading socfpga.dtb

15576 bytes read
reading soc_system.rbf

7007184 bytes read
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:   Linux-3.8.0-xillinux-1.1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3328336 Bytes = 3.2 MiB
   Load Address: 00008000
   Entry Point:  00008000
## Flattened Device Tree blob at 00000100
   Booting using the fdt blob at 0x00000100
   XIP Kernel Image ... OK
OK
   Loading Device Tree to 0fff8000, end 0fffecd7 ... OK

Starting kernel ...
```

この時点で、 boot loader は Linux kernelに制御を渡します。 kernelの boot メッセージの冒頭のみを以下に示します。 kernel が boot シーケンスを終了した後、 shell prompt が与えられます。

“Starting kernel...” の後に何も表示されない場合は、青色の LED がボード上で点滅していることを確認してください。これは、 FPGA パーツが正常にロードされたことを示しています。 soc_system.rbf ファイルのロードに失敗したことが、この原因である可能性が最も高いです。

```
Booting Linux on physical CPU 0x0
Initializing cgroup subsys cpuset
Linux version 3.8.0-00116-gffe44a8 (eli@ocho.localdomain) (gcc version 4.6.3 (S3
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=10c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Altera SOCFPGA, model: Altera SOCFPGA Cyclone V
Memory policy: ECC disabled, Data cache writealloc
PERCPU: Embedded 8 pages/cpu @80e77000 s10880 r8192 d13696 u32768
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 260096
Kernel command line: console=ttyS0,57600 root=/dev/mmcblk0p3 rw rootwait
```

4.5 最初の bootの直後に行う

4.5.1 file systemのサイズを変更します

root file system イメージは小さく保たれるため、デバイスへの書き込みは可能な限り高速になります。一方、 microSD 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、 *microSD* カードの内容全体が消去される重大なリスクがあります。したがって、できるだけ早くこれを行うことをお勧めしますが、そのような事故の代償は単に *microSD* カードの初期化 *(*イメージと *soc_system.rbf*の書き込み*)* を繰り返すことです。

開始点は通常、次のとおりです。

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       2.3G  1.9G  304M  87% /
devtmpfs        505M  4.0K  505M   1% /dev
none            101M  736K  101M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            505M     0  505M   0% /run/shm
```

したがって、 root filesystem は 2.3 GBで、304 MB が空いています。

最初の段階は、 microSD カードを再分割することです。 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
- 3 [ENTER] – partition 番号 3 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 3、可能な限り低い sector から始まり、可能な限り高いもので終わります。

- 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。 microSD 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。

```
root@localhost:~# fdisk /dev/mmcblk0

Command (m for help): d
Partition number (1-4): 3

Command (m for help): n
Partition type:
   p   primary (2 primary, 0 extended, 2 free)
   e   extended
Select (default p):
Using default response p
Partition number (1-4, default 3):
Using default value 3
First sector (112455-15523839, default 112455):
Using default value 112455
Last sector, +sectors or +size{K,M,G} (112455-15523839, default 15523839):
Using default value 15523839
```

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16:
         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)
Syncing disks.
```

システムに表示される最初の sector のデフォルトが上記のものと異なる場合は、ここに表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。

file system を可能な限り小さくするために、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、最後の sectorです。

下部の警告にあるように、 root partition が使用中のため、 partition table の Linuxのビューは更新できません。したがって、再起動が必要です。

```
# shutdown -h now
```

UART console に “System halted.” というメッセージが表示された後 (または VGA 画面でカーソルの点滅が停止した後)、ボードの電源をオフにしてから再度オンにします。システムは以前と同じように boot を実行する必要がありますが、再パーティション化中に何かが正しく行われなかった場合、 boot はどの段階でも失敗する可能性があります。

file system はまだサイズ変更されていません。サイズを変更する余地しか与えられていません。したがって、 shell promptで、次の応答が期待される

```
# resize2fs /dev/mmcblk0p3
```

と入力します。

```
resize2fs 1.42 (29-Nov-2011)
Filesystem at /dev/mmcblk0p3 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mmcblk0p3 is now 1926423 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 microSD 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  1.9G  5.0G  28% /
devtmpfs        505M  4.0K  505M   1% /dev
none            101M  736K  101M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            505M     0  505M   0% /run/shm
```

“df -h” ユーティリティによって指定されるサイズは、10^9 bytesの Gigabyte よりも 7.3% 大きい 1 GiB = 2^{30} bytesであることに注意してください。 8 GB カードが 7.1 GiB として表示されるのはそのためです。

4.5.2 リモート SSH アクセスを許可する

ボードに ssh サーバーをインストールするには、ボードをインターネットに接続し、 shell promptに

```
# apt-get install ssh-server
```

と入力します。 root のパスワードはデフォルトでは何も設定されておらず、 ssh はパスワードなしで誰かがログインすることを正当に拒否することに注意してください。

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
```

SSH サーバーをインストールした後、 /etc/ssh/sshd_configの下部に次の行を追加することをお勧めします。

```
UseDNS no
```

これにより、 SSH サーバーによって行われるかなり無意味な逆 DNS チェックがオフになり、セッションを開始しようとすると遅延が発生します。

4.6 デスクトップの使用

Xillinux デスクトップは、他の Ubuntu デスクトップと同じです。 microSD 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読み込みはやや遅くなりますが、デスクトップ自体はかなり反応します。

デスクトップ環境でアプリケーションを実行するには、デスクトップの左上にある Ubuntu アイコン (“Dash home”) をクリックし、目的のアプリケーションの名前を入力します (例: shell prompt terminal ウィンドウの場合は “terminal” 、gedit テキストエディターの場合は “edit”)。

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。

4.7 シャットダウン中

システムの電源を切るには、デスクトップの右上のアイコンを選択し、 “Shut Down...”をクリックします。または、 shell promptに

```
# shutdown -h now
```

と入力します。

“System Halted” というテキスト メッセージが UART consoleに表示されたら、ボードの電源を切っても安全です。または、 cursor が VGA 画面のテキスト console で点滅を停止した場合、それは Linux がシャットダウンしたことのサインでもあります。

4.8 ここから何をすべきか

SoCKit 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。

Xillinux には gcc compiler と GNU makeが含まれているため、 host アプリケーションはボードの processorsで直接 compiled にすることができます。 apt-get を使用して、追加のパッケージをディストリビューションに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

重要:

FPGA design が *demo bundle*で提供されたもの以外の *QSF* ファイルで構築されている場合、詳細については *5.3* の段落を参照してください。

Xillinux ディストリビューションは、 application logicと簡単に統合できるようにセットアップされています。データ ソースと sinks 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 image キットの他のすべての 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 3.4で処理されたプロジェクトに追加し、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。 logicが更新されたシステムの boot の場合、 soc_system.rbf は 3.4 の説明に従って再生成され、 3.6の説明に従って microSD カードに書き込まれる必要があります。 最初のディストリビューション展開の他の手順を繰り返す必要がないため、 logic の開発サイクルは非常に迅速かつ単純です。

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。

これに対する例外は、メモリまたは register アレイを Xillybus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 から到着したデータをループバックします。 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 sinkとして機能します。

より有用なシナリオでは、 FIFOの一方の端だけが 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 sink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で駆動されるため、両側に 1 つの共通 clock のみを受け入れます。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 sinks 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への streams に対して (First Word Fall Throughとは対照的に)単純な FIFO インターフェース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

SoCKit ボード以外のボードで 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。

とりわけ、これらにはピンと clocksの交換が含まれます。

- xillybus.v で定義された clock PLL の属性は、 clk_bot1に接続されているフリーランニング clock と一致するように設定する必要があります (それが 50 MHz でない場合) (それに応じて SDC ファイルが更新されます)。
- VGA の出力は、目的のボードに一致する必要があります。
- HPSの多重化されたピン: ARM core には、固定配置でチップ上の物理ピンにルーティングされる I/O ピンがあります。 ARM core は Qsys 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。

後者の問題は重要です。重要なハードウェア機能が失敗する可能性があるだけでなく、ボード上の不正な物理信号条件がハードウェアに損傷を与える可能性があるためです (実際の損傷は非常にまれですが)。

HPS のピン割り当ての不一致は、次の 3 つの手順で修正されます。

- Qsysでこれらの割り当てを修正します。
- プロジェクト全体 (FPGA 部分を含む) をビルドし、新しい handoff ファイルに基づいて更新された boot loader を生成します (processorで関連する registers をセットアップするのは U-boot の SPL 部分です)。
- デバイスの割り当てに一致するように DTS ファイルを更新し、このファイルの compilation を実行して、システムの boot を実行するための DTB ファイルを生成します。

5.3 カスタム ビルド プロジェクトと preflow.tcl

demo bundle のソースが別の Quartus II プロジェクトに採用される場合、 QSF ファイル間で情報を転送するための一般的な方法が適用されます。 demo bundleの soc_system サブディレクトリに存在する非標準の script、 preflow.tclを処理するには、特別な注意が必要です。

scriptの主な目的は、相互接続を実装するバグのある AXI bus logic をバイパスするために、 Qsysによって生成された SoC インフラストラクチャの toplevel module を再配線することです。この再配線がなければ、 AXI bus を介したデータ転送は適切に機能しているように見えますが、散発的なデータ破損が観察されます。

script は、 compilation が実行される前に Quartus II によって自動的に実行されます。これは、 xillydemo.qsfの次の行の結果です。

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE \
                  quartus_sh:../soc_system/preflow.tcl
```

Xillybusベースの FPGA design が再配線されていないインフラストラクチャを誤って使用しないようにするために、 preflow.tcl は一部の top-level ポートの名前も変更し、元のポートと互換性がないようにします。上記のポートは、 xillybus_0_conduit_*という名前のポートです。

モジュールが scriptによって変更されたかどうかは簡単にわかります。ファイルが実際に再配線されている場合、変更されたファイルの最初の行 soc_system/soc_system/synthesis/so には

```
// soc_system.v (mangled by preflow.tcl)
```

が含まれています。

カスタム プロジェクトに SoC インフラストラクチャを含めるには、既に完全にビルドされた demo bundleからファイルをコピーして SoC モジュール ツリー全体を採用し、次の行を QSF ファイルに追加します。

```
set_global_assignment -name QIP_FILE path/to/soc_system.qip
```

5.4 Qsys プロジェクトの変更点

Qsysによって生成された Verilog ファイルの 1 つが自動的に変更されるため、 Qsys プロジェクトを変更することはお勧めしません。特にプロジェクトのトポロジが変更された場合、変更された Qsys プロジェクトが scriptによって正しく修正されるという保証はありません。

processorのピン多重化の変更など、要素の属性の変更はおそらく問題ありません。ただし、 HPS processorの属性が変更された場合、 logicではなく、プリローダーに含まれる C ファイルの変更によって有効になることに注意してください。

6

トラブルシューティング

6.1 ポート「xillybus_0_conduit_...」が存在しません

Quartus IIを使用した Xillydemo プロジェクトの compilation 中に、次のようなエラー メッセージが表示される場合があります。

```
Error (12002): Port "xillybus_0_conduit_M_AXI_ARADDR" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARBURST" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARCACHE" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARID" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARLEN" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARLOCK" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARPROT" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARREADY" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARSIZE" does not exist in macrofunction "u0" File: xillybus.v Line: 442
...
```

ポートが見つからないということは、 compilation より前に実行されるはずだった script が実行されなかったことを示している可能性があります。これは、 QSF ファイルの誤った変更、またはカスタム Quartus II プロジェクトへのソースの不適切な採用の結果である可能性があります。詳細については、段落 5.3を参照してください。

6.2 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。

- Linux が bootを実行したとき、デバイスは接続されていましたか?そうでない場合は、マウスやキーボードを接続した状態で Linux を再起動します。

- 正しい USB プラグを使用していますか?中央にある “HPS USB”とマークされたものである必要があります。
- USB hub を使用する場合は、 SoCKit ボードの OTG ポートに接続されている USB ケーブルに、キーボードまたはマウスのみを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は serial terminal でログを表示できます。 UART リンクのセットアップ方法については、 SoCKit ボードのドキュメントを参照してください。

6.3 File system マウントの問題

経験によれば、適切な microSD 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、永続ストレージにまったく問題はありません。

ext4 file system は次の mountで journal を使用して修復されるため、 root file system をアンマウントせずにボードの電源をオフにしても、 file system 自体に永続的な不一致が生じる可能性は低いです。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が誤った内容のままになったり、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 システムの機能に蓄積された損傷があります。これは、突然電源がオフになったコンピューターに当てはまります。

root file system がマウントに失敗する (boot中に kernel panic になる) か、 mount 読み取り専用を実行する場合、最も可能性の高い原因は低品質の microSD カードです。このようなストレージがしばらくの間正常に機能した後、ランダムなエラーメッセージが表示され始めるのは非常に一般的なことです。 /var/log/syslog に次のようなメッセージが含まれている場合、 (Micro)SD カードが原因である可能性が最も高くなります。

```
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。

6.4 “startx” が失敗する (グラフィカル デスクトップが起動しない)

直接関係はありませんが、 microSD 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 microSD 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 microSD カードを使用することです。