(기계로 한국어 번역)

Getting started with Xillinux for Zynq-7000 v2.0

Xillybus Ltd.

www.xillybus.com

Version 4.2

이 문서는 영어에서 컴퓨터에 의해 자동으로 번역되었으므로 언어가 불분명할 수 있습니다. 이 문서는 원본에 비해 약간 오래되었을 수 있습니다.

가능하면 영문 문서를 참고하시기 바랍니다.

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 This document may also be slightly outdated in relation to the original.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

소개

1.1 Xillinux 배포판

Xillinux는 Zynq-7000 장치를 위한 완전한 그래픽 Lubuntu 16.04 기반 Linux 배포판으로, 혼합 소프트웨어/logic 프로젝트의 신속한 개발을 위한 플랫폼으로 고안되었습니다. 현재 지원되는 보드는 Z-Turn Lite, Zedboard, MicroZed 및 Zybo입니다.

다른 Linux 배포판과 마찬가지로 Xillinux는 Linux를 실행하는 개인용 데스크탑 컴퓨터와 거의 동일한 기능을 지원하는 소프트웨어 모음입니다. 일반적인 Linux 배포판과 달리 Xillinux에는 일부 하드웨어 logic, 특히 VGA 어댑터도 포함되어 있습니다.

Z-Turn Lite, Zedboard 및 Zybo를 사용하면 배포판이 클래식 키보드, 마우스 및 모니터 설정에 맞게 구성됩니다. 또한 USB UART 포트에서 명령줄 제어를 허용하지만 이 기능은 주로 문제 해결에 사용할 수 있습니다.

VGA/DVI 출력이 없는 MicroZed와 함께 사용하면 USB UART만 console로 사용됩니다.

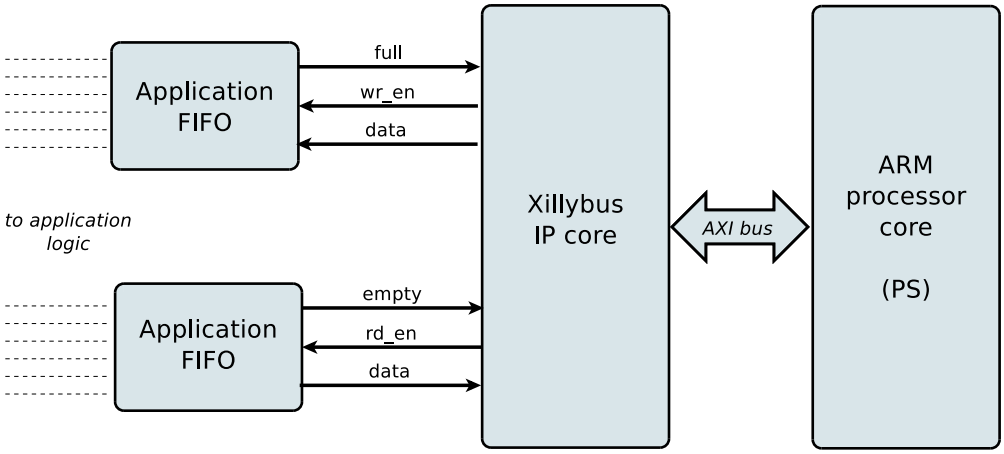
Xillinux는 또한 장치의 FPGA logic fabric와 ARM processor에서 실행되는 일반 user space applications 간의 통합을 위한 킥스타트 개발 플랫폼입니다. 포함된 Xillybus IP core 및 driver를 사용하면 FPGA logic 및 Linux 기반 소프트웨어가 함께 작동하는 애플리케이션의 design을 완성하는 데 프로그래밍 및 logic design의 기본 기술만 있으면 됩니다.

번들로 제공되는 Xillybus IP cores는 애플리케이션 설계자에게 간단하면서도 효율적인 작업 환경을 제공함으로써 application logic와 processor 간의 인터페이스는 물론 kernel programming의 하위 수준 내부를 처리할 필요가 없습니다.

1.2 Xillybus IP core

Xillybus는 FPGA와 Linux 또는 Microsoft Windows를 실행하는 host 간의 데이터 전송을 위한 DMA 기반 종단 간 솔루션입니다. FPGA logic의 설계자와 소프트웨어 프로그래머에

게 간단하고 직관적인 인터페이스를 제공합니다. PCI Express bus를 기본 전송으로 사용하는 개인용 컴퓨터 및 embedded 시스템과 AMBA bus(AXI3/ AXI4)와 인터페이스하는 ARM 기반 processors에서 사용할 수 있습니다.



위에 표시된 것처럼 FPGA의 application logic은 표준 FIFOs와만 상호 작용하면 됩니다.

예를 들어, 다이어그램에서 하위 FIFO에 데이터를 쓰는 것은 Xillybus IP core가 FIFO의 다른 쪽 끝에서 데이터를 전송할 수 있음을 감지하게 합니다. 곧 IP core는 FIFO에서 데이터를 읽고 host로 보내 userspace software에서 읽을 수 있도록 합니다. 데이터 전송 메커니즘은 FIFO와만 상호 작용하는 FPGA의 application logic에 투명합니다.

반면 Xillybus IP core는 AXI bus를 활용하여 데이터 흐름을 구현하여 processor core의 bus에서 DMA 요청을 생성합니다.

host의 응용 프로그램은 named pipes처럼 동작하는 device files와 상호 작용합니다. Xillybus IP core 및 driver는 FPGAs의 FIFOs와 host의 관련 device files 간에 데이터를 효율적이고 직관적으로 전송합니다.

IP core는 온라인 웹 애플리케이션을 사용하여 고객의 사양에 따라 즉시 구축됩니다. streams의 수, 방향 및 기타 속성은 design의 대역폭 성능, 동기화 및 단순성 간의 최적 균형을 달성하기 위해 고객이 정의합니다.

이 가이드에 설명된 대로 준비를 마친 후 http://xillybus.com/custom-ip-factory에서 사용자 지정 IP core를 빌드하고 다운로드하는 것이 좋습니다.

이 가이드는 Xillybus IP core가 포함된 Xillinux 배포판을 빠르게 설정하는 방법을 설명합니다. 이 IP core는 실제 애플리케이션 시나리오 테스트를 위해 사용자 제공 데이터 소스 및 데이터 소비자에 연결할 수 있습니다. 데모 키트가 아니라 유용한 작업을 있는 그대로 수행할 수 있는 완전한 기능의 starter design입니다.

기존 IP core를 특수 응용 프로그램에 맞게 조정된 것으로 교체하는 것은 빠른 프로세스이

며 하나의 바이너리 파일을 교체하고 하나의 단일 모듈을 인스턴스화해야 합니다.

Xillybus IP core 사용에 대한 자세한 내용은 다음 문서에서 찾을 수 있습니다.

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

궁금하신 분들을 위해 Xillybus IP core의 구현 방법에 대한 간략한 설명은 Xillybus host application programming guide for Linux의 부록 A에서 찾을 수 있습니다.

2

전제 조건

2.1 하드웨어

Zynq용 Xillybus의 Linux 배포판인 Xillinux는 현재 다음 보드를 지원합니다.

- Z-Turn Lite(사용 가능한 모든 Zynq 장치와 함께)
- Zedboard
- 7010 MicroZed. 7020 MicroZed는 사소한 수정으로도 지원됩니다. 섹션 5.5을 참조하십시오.
- Zybo

Z-Turn Lite 보드를 구입하려는 경우 이 보드의 web page를 방문하여 항목 선택에 대한 도움을 받는 것이 좋습니다.

위에 나열되지 않은 보드의 소유자는 자신의 하드웨어에서 배포판을 실행할 수 있지만 일부는 중요하지 않을 수 있는 특정 변경이 필요할 수 있습니다. 이에 대한 자세한 내용은 5.2 섹션을 참조하십시오.

보드(MicroZed 제외)를 모니터, 키보드 및 마우스가 있는 데스크탑 컴퓨터로 사용하려면 다음 항목이 필요합니다.

- 보드의 출력(즉, 거의 모든 PC 모니터)에 따라 아날로그 VGA 또는 HDMI 입력으로 VESA 호환 1024x768 @ 60Hz를 표시할 수 있는 모니터.
- 모니터용 VGA 또는 HDMI 케이블(해당되는 경우)
- USB 키보드
- USB 마우스

- USB hub(키보드와 마우스가 단일 USB 플러그에 결합되지 않은 경우)

Z-Turn Lite 보드에는 모니터에 대한 출력이 없습니다. 따라서 HDMI 포트가 있는 Z-Turn Lite IO Cape board를 데스크탑 사용 시나리오에 연결해야 합니다.

Zybo를 사용하는 경우 모니터를 HDMI 포트와 VGA 포트에 연결할 수 있습니다. Zedboard의 HDMI 출력 포트는 지원하지 않습니다.

무선 키보드/마우스 콤보는 USB hub의 필요성을 제거하고 실수로 USB 케이블을 잡아당겨 보드의 USB 포트에 물리적 손상을 줄 수 있으므로 권장됩니다.

Zedboard 및 Z-Turn Lite에서 키보드와 마우스의 연결은 Micro B - Type A 암 USB 케이블을 통해 이루어지며 Zedboard 및 가능하면 Z-Turn Lite와 함께 제공됩니다(구입 품목에 따라 다름).

다른 두 보드에서는 표준 USB 유형 A 암 커넥터(PC의 USB 플러그와 같은)를 주변 장치 연결에 사용할 수 있습니다.

또한 필수 사항:

- 4GB 이상을 지원하는 안정적인 SD 카드(Zedboard용) 또는 MicroSD(Z-Turn Lite, MicroZed 및 Zybo용), 가장 바람직하게는 Sandisk에서 제조한 카드입니다. (아마도) 보드와 함께 제공된 카드는 Xillinux에서 문제가 보고되었기 때문에 권장하지 않습니다.
- 권장: image 및 boot file을 카드에 쓰기 위한 (Micro)SD 카드와 PC 사이의 USB 어댑터. PC 컴퓨터에 SD 카드용 내장 슬롯이 있는 경우 이 작업이 필요하지 않을 수 있습니다. Zynq 보드 자체도 SD 카드에 쓰는 데 사용할 수 있지만 이것은 다소 어렵습니다.

2.2 배포판 다운로드

Xillinux 배포판은 Xillybus 사이트의 다운로드 페이지에서 다운로드할 수 있습니다.

http://xillybus.com/xillinux/

배포판은 두 개의 개별 파일로 다운로드되는 두 부분으로 구성됩니다.

- 부팅 시 Linux에서 볼 수 있는 file system로 구성된 (Micro)SD 카드의 raw image
- boot partition을 채우기 위해 Xilinx의 도구로 implementation을 실행하기 위한 파일 세트인 boot partition kit.

이에 대한 자세한 내용은 3 섹션을 참조하십시오.

배포판에는 processor와 logic fabric 간의 손쉬운 통신을 위한 Xillybus IP core 데모가 포함되어 있습니다. 이 demo bundle의 특정 구성은 간단한 테스트를 위한 것이므로 특정 응용 프로그램에서 상대적으로 성능이 저하될 수 있습니다.

사용자 지정 IP cores는 IP Core Factory 웹 애플리케이션을 사용하여 구성, 자동 구축 및 다운로드할 수 있습니다. 이 도구를 사용하려면 http://xillybus.com/custom-ip-factory를 방문하십시오.

Xillybus IP core 및 Xillinux 배포를 포함하여 다운로드한 모든 번들은 이 사용이 “evaluation”라는 용어와 합리적으로 일치하는 한 무료로 사용할 수 있습니다. 여기에는 최종 사용자 designs에 IP core 통합, 실제 데이터 실행 및 현장 테스트가 포함됩니다. IP core 사용의 유일한 목적이 특정 애플리케이션에 대한 기능과 적합성을 평가하는 것이라면 IP core 사용 방법에는 제한이 없습니다.

2.3 개발 소프트웨어

Vivado 2014.4 **이상** 은 Xillinux 배포판에서 logic design의 compilation에 사용할 수 있습니다.

7z007s 또는 7z014s 장치를 사용하려는 경우 이를 지원하는 Vivado 개정판이 필요합니다(예: Vivado 2016.4 이상).

소프트웨어는 Xilinx의 웹사이트(http://www.xilinx.com)에서 직접 다운로드할 수 있습니다.

다음을 포함한 모든 Vivado 에디션이 적합합니다.

- WebPack Edition을 다운로드하여 라이선스 비용 없이 무제한으로 사용할 수 있습니다. Z-Turn Lite, Zedboard, MicroZed 및 Zybo와 함께 제공되는 모든 장치는 이 에디션의 적용을 받습니다.
- 구매한 라이선스가 필요한 Design Edition(30일 평가판 사용 가능).
- 구입한 보드와 함께 특별히 라이선스가 부여되었을 수 있는 모든 에디션, 따라서 특정 Zynq 장치로 제한됩니다.
- System 및 WebPack Edition의 상위 기능 집합을 제공하는 다른 에디션도 괜찮습니다.

이 모든 에디션은 추가 라이선스 없이 Zynq용 Xillybus를 구현하는 데 필요한 Xilinx 제공 IP cores를 다룹니다.

2.4 FPGA design 사용 경험

Z-Turn Lite, Zedboard, MicroZed 또는 Zybo를 사용할 때 플랫폼에서 배포판을 실행하기 위해 FPGA design에 대한 이전 경험이 필요하지 않습니다. 다른 보드를 사용하려면 Xilinx의 도구 사용에 대한 약간의 지식이 필요하고 Linux kernel와 관련된 몇 가지 기본 기술이 필요할 수 있습니다.

배포판을 최대한 활용하려면 logic design 기술을 잘 이해하고 HDL 언어(Verilog 또는 VHDL)를 마스터해야 합니다. 그럼에도 불구하고 Xillybus 배포판은 실험할 간단한 스타터 design을 제공하므로 이러한 학습을 위한 좋은 출발점입니다.

3

건물 Xillinux

3.1 개요

Xillinux 배포판은 단순한 데모가 아닌 개발 플랫폼으로 고안되었습니다. 맞춤형 logic 개발 및 통합을 위한 즉시 사용 가능한 환경은 하드웨어에서 실행하기 위한 준비 과정에서 구축됩니다. 따라서 첫 번째 테스트 실행을 위한 준비 시간은 약간 길다(일반적으로 30분, 대부분은 Xilinx의 도구를 기다리는 시간으로 구성됨). 그러나 이러한 긴 준비는 맞춤형 logic 통합 주기를 단축시킵니다.

(Micro)SD 카드에서 Xillinux 배포의 성공적인 boot 프로세스를 위해서는 두 가지 구성 요소가 있어야 합니다.

- boot partition의 FAT32 filesystem은 boot loaders, FPGA 부품용 configuration bitstream(PL로 알려짐), Linux kernel의 boot용 바이너리 파일로 구성됩니다.
- Linux에 탑재된 ext4 root file system.

다운로드한 Xillinux의 raw image에는 이미 거의 모든 것이 설정되어 있습니다. boot partition에는 세 개의 파일만 누락되어 있으며 그 중 하나는 Xilinx의 도구로 생성해야 하고 두 개는 boot partition kit에서 복사해야 합니다.

(Micro)SD 준비를 위한 다양한 작업은 이 섹션에서 단계별로 자세히 설명합니다.

이 절차는 아래에 설명된 순서대로 수행해야 하는 다음 단계로 구성됩니다.

- boot partition kit 압축 풀기
- 메인 PL(FPGA) 프로젝트의 implementation 실행
- Xillinux image를 (Micro)SD 카드에 쓰기
- (Micro)SD 카드의 boot partition에 3개의 파일 복사

다른 보드와 작업하는 방법은 5.2 단락에서 설명합니다.

3.2 boot partition kit 압축 풀기

이전에 다운로드한 xillinux-eval-*board-XXX*.zip 파일을 작업 디렉토리에 압축을 풉니다.

중요한:

작업 디렉토리의 경로에는 공백이 포함되지 않아야 합니다. 특히 Windows Desktop은 경로에 "Documents and Settings"가 포함되어 있으므로 적합하지 않습니다.

번들은 다음 디렉토리(또는 그 중 일부)로 구성됩니다.

- verilog– 기본 logic에 대한 프로젝트 파일과 Verilog의 일부 소스('src' 하위 디렉토리에 있음)를 포함합니다.
- vhdl– 기본 logic 및 일부 소스 파일에 대한 프로젝트 파일을 포함합니다. VHDL에서 편집할 파일은 'src' 하위 디렉토리에 있습니다.
- cores– Xillybus IP cores의 미리 컴파일된 바이너리
- system– processor 관련 logic을 생성하기 위한 디렉토리
- bootfiles– boot partition에 복사할 두 개의 보드별 파일이 포함되어 있습니다.
- vivado-essentials– Vivado에서 사용하기 위한 processor 관련 및 범용 logic에 대한 정의 파일 및 빌드 디렉토리.

Z-Turn Lite, Zedboard, MicroZed 및 Zybo 보드에 사용할 수 있는 번들이 있습니다. 다른 보드를 사용하는 경우 5.2 섹션에 나열된 문제 외에도 constraints 파일 vivado-essentials/xillydemo.xdc를 적절하게 편집해야 합니다.

vhdl 디렉토리에는 Verilog 파일이 포함되어 있지만 사용자가 편집할 필요는 없습니다.

Xillybus IP core와의 인터페이스는 각 'src' 하위 디렉토리의 xillydemo.v 또는 xillydemo.vhd 파일에서 발생합니다. 이것은 자신의 데이터 소스 및 데이터 소비자로 Xillybus를 시도하기 위해 편집할 파일입니다.

3.3 bitstream 파일 생성

3.3.1 소개

Vivado는 비교적 복잡한 구조에서 많은 중간 파일을 생성하므로 프로젝트를 제어하기가 어렵습니다. 번들의 파일 구조를 컴팩트하게 유지하기 위해 Vivado 프로젝트 생성을 위해

Tcl의 script가 제공됩니다. 이 script는 새 하위 디렉토리 “vivado”를 만들고 필요에 따라 이 디렉토리를 파일로 채웁니다.

프로젝트는 src/ 하위 디렉토리의 파일에 의존합니다(이 파일의 복사본은 만들어지지 않음). processor, 상호 연결 및 주변 장치, logic에서 사용하는 FIFOs는 vivado-essentials/에 정의되어 있으며 프로젝트의 implementation 동안 Vivado에 의해 중간 파일도 채워집니다.

프로젝트의 implementation은 Verilog 또는 VHDL을 기반으로 할 수 있습니다.

3.3.2 원하는 Zynq 부품 선택(Z-Turn Lite만 해당)

이 단계는 Z-Turn Lite용 demo bundle **에서만** 필요합니다.

텍스트 편집기를 사용하여 번들의 root directory에서 select_part.tcl을 엽니다. 이 파일의 마지막 네 줄은 Vivado 프로젝트가 생성되는 Zynq 부분을 설정하기 위한 Tcl 명령입니다. 이 네 줄은 “#” 문자로 주석 처리됩니다.

Z-Turn Lite 보드에서 Zynq 부품을 선택하려면 이 행 중 하나에서 “#” 문자 하나의 주석 처리를 제거하십시오.

나중에 다른 Zynq 부품을 사용하려면 Vivado 프로젝트의 설정을 적절히 변경하고 프로젝트를 다시 구현하십시오. select_part.tcl은 프로젝트 생성 중에만 참조되므로 나중에 변경해도 영향이 없습니다.

3.3.3 xillydemo.vhd 준비 중(VHDL 프로젝트만 해당)

이 단계는 다음과 같은 경우에만 필요합니다.

- 프로젝트는 VHDL에 있습니다.
- demo bundle은 Z-Turn Lite를 **제외한** 모든 보드용입니다.

두 조건이 모두 충족되면 vhdl/src/xillydemo.vhd를 편집해야 합니다. Xillydemo의 엔티티 포트 목록 시작 부분에서 다음 세 줄을 **제거합니다** .

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

또한 아키텍처 정의에서 다음 줄의 주석을 **제거하십시오** (“--” 주석 표시 제거).

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

Verilog 소스 파일을 변경할 필요가 없습니다.

3.3.4 Vivado 프로젝트 생성

Vivado 2014.4 이상을 시작합니다.

프로젝트가 열려 있지 않은 상태에서 Tools > Run Tcl Script...를 선택하고 기본 설정에 따라 verilog/ 또는 vhdl/ 하위 디렉터리에서 **xillydemo-vivado.tcl**을 선택합니다. 일련의 이벤트가 1분 이내에 발생합니다. 프로젝트 배포의 성공 여부는 Vivado 창 하단에 있는 “Tcl Console” 탭을 선택하여 확인할 수 있습니다.

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

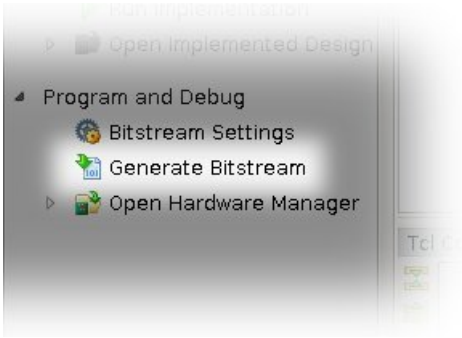
이것이 Tcl console 출력의 마지막 줄이 아닌 경우 문제가 발생한 것입니다.

이 단계에서 Critical Warnings가 있을 수 있지만 오류는 없습니다. 그러나 프로젝트가 이미 생성된 경우(즉, script가 이미 실행된 경우) script를 다시 실행하려고 하면 다음 오류가 발생합니다.

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

3.3.5 프로젝트의 Implementation

프로젝트가 생성된 후 implementation을 실행합니다. 왼쪽에 있는 Flow Navigator bar에서 “Generate Bitstream”을 클릭합니다.



synthesis 및 implementation을 실행해도 되는지 묻는 팝업 창이 나타나면 “Yes”를 선택합니다.

Vivado는 일련의 프로세스를 실행합니다. 이 작업은 일반적으로 몇 분 정도 걸립니다. 여러 warnings가 출시되었으며 그 중 일부는 위험으로 분류될 수 있습니다. 오류가 없어야 합니다.

bitstream이 성공적으로 생성되었음을 알리는 팝업 창이 나타나 다음에 수행할 작업을 선택할 수 있습니다. “Cancel” 선택을 포함하여 모든 옵션이 좋습니다.

bitstream 파일 xillydemo.bit은 vivado/xillydemo.runs/impl_1/에서 찾을 수 있습니다.

implementation은 결코 실패할 것으로 예상되지 않습니다. 그러나 언급할 가치가 있는 몇 가지 오류 조건이 있습니다.

- VHDL design에서 “IO placement is infeasible”와 함께 Placer가 실패합니다. VHDL이 있는 implementation에서 이런 일이 발생하면 위에서 요구한 대로 xillydemo.vhd가 편집되었는지 확인하십시오.
- write_bitstream은 DRC 오류로 실패하고 PS_CLK, PS_PORB 및 PS_SRSTB가 지정되지 않고, 라우팅되지 않고, 제약이 없다고 불평합니다. 그런 다음 다시 – xillydemo.vhd가 위에서 요구한 대로 편집되었는지 확인하십시오.
- “Timing constraints weren’t met”을 말하는 동안 오류가 발생했습니다. 이는 사용자 지정 logic이 통합되어 도구가 timing 요구 사항을 충족하지 못하는 경우 발생할 수 있습니다. 이는 design이 구문적으로 정확하지만 주어진 clock 속도 및/또는 I/O 요구 사항과 관련하여 특정 경로를 충분히 빠르게 만들기 위해 수정이 필요함을 의미합니다. 더 나은 timing을 위해 design을 수정하는 프로세스를 종종 *timing closure*라고 합니다.

  timing constraint 오류는 일반적으로 critical warning로 발표되어 사용자가 FPGA의 동작이 보장되지 않는 bitstream 파일을 생성할 수 있습니다. 이러한 bitstream의 생성을 방지하기 위해 timing 오류는 route 실행이 끝날 때 자동으로 실행되는 작은 Tcl script, “showstopper.tcl” 덕분에 오류 수준으로 승격됩니다. 이 안전 조치를 끄려면 Flow Navigator의 “Project Manager” 아래에 있는 “Project Settings”를 클릭하십시오. “Implementation” 버튼을 선택하고 “route_design”에 대한 설정까지 아래로 스크롤합니다. 그런 다음 tcl.post에서 showstopper.tcl을 제거합니다.
- 다른 모든 오류는 사용자가 변경한 결과일 가능성이 높으므로 사례별로 처리해야 합니다.

3.4 image와 함께 (Micro)SD 로드

3.4.1 일반적인

이 작업의 목적은 image file을 (micro)SD 카드에 쓰는 것입니다. 이 파일의 이름은

xillinux-2.0a.img.gz이며 압축 파일(gzip 형식)로 다운로드됩니다.

이 (Micro)SD 카드의 image는 boot에 사용할 준비가 되어 있으며, 카드에 쓴 후 추가되는 3개의 파일을 제외합니다.

이 image는 압축을 풀고 (Micro)SD 카드의 첫 번째 sector 이상에 기록되어야 합니다. 이를 수행하는 몇 가지 방법과 도구가 있습니다. 몇 가지 방법이 다음에 제안됩니다.

image에는 partition table, 초기 boot files를 배치하기 위한 부분적으로 채워진 FAT file system 및 ext4 유형의 Linux root file system이 포함되어 있습니다. 두 번째 partition은 거의 모든 Windows 컴퓨터에서 무시되므로 (Micro)SD 카드는 용량이 매우 작은 것처럼 보일 수 있습니다(16 MB 정도).

full disk image를 작성하는 것은 일반 컴퓨터 사용자를 위한 작업이 아니므로 Windows 컴퓨터에서는 특별한 소프트웨어가 필요하고 Linux에서는 특별한 주의가 필요합니다. 다음 단락에서는 두 운영 체제에서 이 작업을 수행하는 방법을 설명합니다.

(Micro)SD 카드용 USB 어댑터(또는 컴퓨터의 전용 슬롯)가 없는 경우 3.4.4 단락에 설명된 대로 보드 자체를 사용하여 image를 작성할 수 있습니다.

중요한:

image를 (Micro)SD에 기록하면 포함되어 있을 수 있는 이전 콘텐츠가 복구 불가능하게 삭제됩니다. image를 작성하는 데 사용된 것과 동일한 도구를 사용하여 기존 콘텐츠의 복사본을 만드는 것이 좋습니다.

3.4.2 image 로드(Windows)

Windows에서 USB Image Tool와 같은 image를 복사하려면 특수 응용 프로그램이 필요합니다. 이 도구는 USB 어댑터를 사용하여 (Micro)SD 카드에 액세스할 때 적합합니다.

일부 컴퓨터(특히 노트북)에는 (Micro)SD 슬롯이 내장되어 있으며 Win32 Disk Imager와 같은 다른 도구를 사용해야 할 수도 있습니다. Windows 7을 실행할 때도 마찬가지입니다.

두 도구 모두 웹의 다양한 사이트에서 무료로 다운로드할 수 있습니다. 다음 연습에서는 USB Image Tool을 사용한다고 가정합니다.

그래픽 인터페이스의 경우 “USB Image Tool.exe”를 실행합니다. 메인 창이 나타나면 USB 어댑터를 연결하고 왼쪽 상단에 나타나는 장치 아이콘을 선택합니다. 왼쪽 상단 drop down menu의 “Device Mode”(“Volume Mode”와 반대)에 있는지 확인하십시오. Restore를 클릭하고 파일 형식을 “Compressed (gzip) image files”로 설정합니다. 다운로드한 image file(xillinux-2.0a.img.gz)를 선택합니다. 전체 프로세스는 약 4-5분이 소요됩니다. 완료되면 장치를 마운트 해제하고(’하드웨어 안전하게 제거’) 플러그를 뽑습니다.

일부 컴퓨터에서 GUI는 소프트웨어 초기화에 실패했다는 오류와 함께 실행에 실패합니다. 이 경우 대체 명령줄을 사용하거나 Microsoft .NET framework 구성 요소를 설치해야 합니다.

또는 명령줄에서 수행할 수 있습니다(GUI 실행 시도가 실패할 경우 빠른 대안). 이것은 두 단계로 수행됩니다. 먼저 장치의 번호를 얻습니다. DOS Window에서 디렉토리를 애플리케이션의 압축을 풀고 이동합니다(일반적인 세션은 다음과 같습니다).

```
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" 뒤의 문자는 숫자 “1”이 아니라 문자 “l”입니다.)

이제 장치 번호가 있으면 실제로 쓰기를 수행할 수 있습니다(’ ’restore”).

```
C:\usbimage>usbitcmd r 2448 \path\to\xillinux-2.0a.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
```

다시 말하지만 이 작업은 약 4-5분 정도 소요됩니다. 그리고 물론, 숫자 2448을 첫 번째 단계에서 얻은 장치 번호로 변경하고 `\path\to`를 (Micro)SD 카드의 image가 컴퓨터에 저장된 경로로 바꾸십시오.

3.4.3 image 로드(Linux)

중요한:

장치에 원시 복사는 위험한 작업입니다. 사람의 사소한 실수(일반적으로 잘못된 대상 디스크 선택)로 인해 컴퓨터 하드 디스크의 모든 데이터가 복구 불가능한 손실로 이어질 수 있습니다. Enter를 누르기 전에 생각하고 Linux에 익숙하지 않은 경우 Windows에서 이 작업을 수행하는 것이 좋습니다.

방금 언급했듯이 올바른 장치를 (Micro)SD 카드로 감지하는 것이 중요합니다. USB 커넥터를 연결하고 기본 로그 파일(/var/log/messages 또는 /var/log/syslog)에서 다음과 같은 항목을 찾는 것이 가장 좋습니다.

```
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
```

출력은 약간 다를 수 있지만 여기서 요점은 kernel이 새 디스크에 어떤 이름을 부여했는지 확인하는 것입니다. 위의 예에서 “sdc”.

image file의 압축을 풉니다.

```
# gunzip xillinux-2.0a.img.gz
```

image를 (Micro)SD 카드로 복사하는 것은 간단합니다.

```
# dd if=xillinux-2.0a.img of=/dev/sdc bs=512
```

물론 플래시 드라이브로 찾은 디스크를 가리켜야 합니다.

중요한:

/dev/sdc를 예로 들어 설명합니다. 컴퓨터에서 인식된 장치와 일치하지 않는 한 이 장치를 사용하지 마십시오.

그리고 확인

```
# cmp xillinux-2.0a.img /dev/sdc
cmp: EOF on xillinux-2.0a.img
```

응답에 유의하십시오. EOF가 image file에서 도달했다는 사실은 다른 모든 항목이 올바르게 비교되었으며 플래시에 실제 사용된 것보다 더 많은 공간이 있음을 의미합니다. cmp가 아무 것도 말하지 않으면(일반적으로 좋은 것으로 간주됨) 실제로 뭔가 잘못되었음을 의미합니다. 장치에 쓰지 않고 일반 파일 "/dev/sdc"가 생성되었을 가능성이 큽니다.

3.4.4 image를 로드하기 위해 Zynq 보드 사용하기

위의 3.4.3 단락은 Linux 머신 및 USB 어댑터로 image를 로드하는 방법을 설명했습니다. Zynq 보드 자체는 보드와 함께 제공된 샘플 Linux 시스템을 실행하여 사용할 수 있습니다. 기본적으로 /dev/mmcblk0을 대상 장치로 사용하여 동일한 지침을 따를 수 있습니다(/dev/sdc 대신).

이것은 boot 프로세스가 QSPI flash에서 수행될 때뿐만 아니라 SD 카드의 샘플 Linux 시스템에서도 잘 작동합니다(보드와 함께 제공된 경우). 이는 Xillinux와 달리 RAM에서 완전히 실행되고 boot가 완료된 후 SD 카드를 사용하지 않기 때문입니다. 따라서 boot용으로 SD 카드를 사용했다면 빼서 image를 쓰기 위해 다른 카드를 삽입할 수 있습니다.

(Micro)SD 및 boot partition 파일의 image에 대한 Zynq 보드 액세스 권한을 부여하는 방법은 선호도와 Linux 지식의 문제입니다. 네트워크를 통해 이를 수행하는 방법에는 여러 가지가 있지만 가장 간단한 방법은 이러한 파일을 디스크 온 키에 쓰고 USB OTG 포트에 연결하는 것입니다. 다음을 사용하여 disk-on-key를 장착합니다.

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

그러면 디스크 온 키의 파일을 /mnt/usb/에서 읽을 수 있습니다.

Zedboard에서 JP2 점퍼가 설치되어 있는지 확인하여 USB 포트에 5V 전원 공급 장치가 공급되도록 합니다.

3.5 boot partition에 파일 복사

이 마지막 단계는 boot에 필요한 파일을 배치합니다.

- boot partition kit의 bootfiles/ 하위 디렉토리에서 boot.bin 및 devicetree.dtb를 (Micro)SD 카드의 boot partition(첫 번째 partition)로 복사합니다.
- 3.3 섹션에서 생성된 xillydemo.bit을 복사합니다(verilog/ 또는 vhdl/ 하위 디렉토리 중 선택된 것에서).

이 파일을 복사하기 전에: (Micro)SD의 image가 카드에 기록된 경우 USB 어댑터를 뽑았다가 다시 컴퓨터에 연결하십시오. Zynq 보드가 raw image 쓰기에 사용된 경우 슬롯에서 (Micro)SD 카드를 당겨서 다시 삽입합니다.

이것은 컴퓨터가 (Micro)SD 카드의 partition table로 최신 상태인지 확인하는 데 필요합니다.

Linux 시스템에서는 첫 번째 partition(예: /dev/sdb1)를 수동으로 마운트해야 할 수도 있습니다. 대부분의 컴퓨터는 이 작업을 자동으로 수행합니다.

예를 들어, Zynq 보드 자체가 이 용도로 사용되는 경우 다음을 입력합니다.

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

파일을 /mnt/sd/에 복사합니다.

Windows 시스템에서 (micro)SD 카드를 연결하면 단일 파일인 uImage가 있는 단일 “disk”이 표시됩니다. 이것은 파일을 복사할 올바른 대상입니다.

완료되면 (Micro)SD 카드를 올바르게 마운트 해제하고 컴퓨터에서 플러그를 뽑습니다.

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

또는 Windows의 “remove the disk safely”.

3.6 boot partition의 파일

boot를 시도하기 전에 boot partition이 다음과 같이 채워져 있는지 확인하십시오.

boot가 성공하려면 (micro)SD 카드의 첫 번째 partition(boot partition)에 4개의 파일이 있어야 합니다.

- uImage– Linux kernel binary. 이것은 Xillinux의 (Micro)SD image를 카드에 쓴 후 boot partition에 있는 유일한 파일입니다. kernel은 보드 독립적입니다.
- boot.bin– 초기 bootloader. 이 파일에는 초기 processor 초기화 및 U-boot 유틸리티가 포함되어 있으며 보드마다 상당히 다릅니다.
- devicetree.dtb– Linux kernel에 대한 하드웨어 정보가 포함된 Device Tree Blob 파일입니다.
- xillydemo.bit– 섹션 3.3에서 생성된 PL (FPGA) 프로그래밍 파일

4

boot 시작

4.1 점퍼 설정

보드가 (Micro)SD 카드에서 boot를 수행하려면 점퍼 설정을 수정해야 합니다. 설정은 아래의 각 보드에 대해 자세히 설명되어 있습니다.

4.1.1 Zedboard

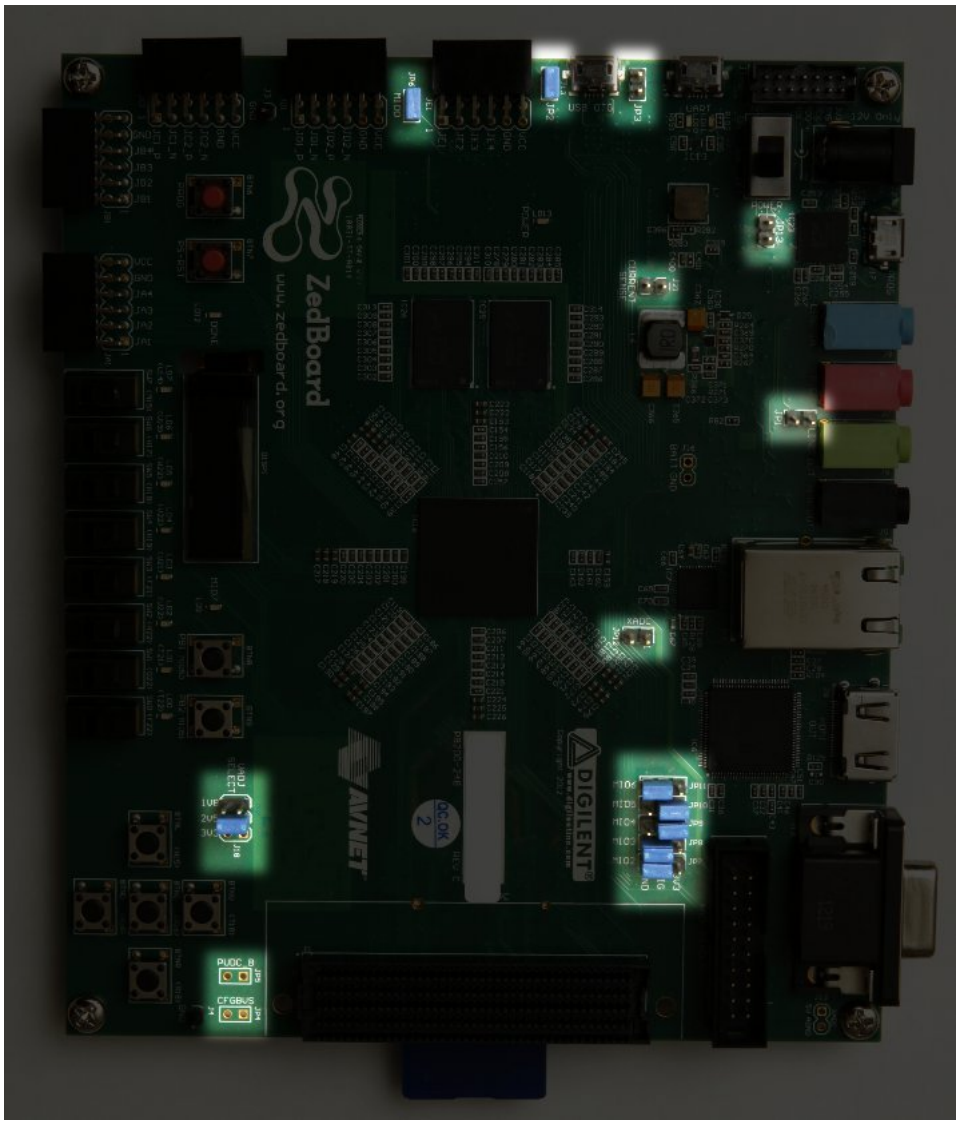
올바른 설정은 다음 페이지의 이미지에 나와 있습니다.

일반적으로 다음과 같은 점퍼 변경이 필요합니다(그러나 처음에는 보드가 다르게 설정될 수 있음).

- 5V를 USB 장치에 공급하려면 JP2용 점퍼를 설치하십시오.
- JP10 및 JP9는 GND에서 3V3 위치로 이동했으며 해당 행의 다른 세 개는 GND에 연결된 상태로 남아 있습니다.
- JP6용 점퍼를 설치합니다(CES silicon에 필요, Zedboard의 Hardware Guide 34페이지 참조).

중요한:

필요한 설정은 JP2가 점퍼된다는 점에서 Zedboard Hardware User Guide에 설명된 설정과 다릅니다. 따라서 보드에 연결된 USB 장치(USB 키보드 및 마우스)는 5V 전원 공급 장치를 받습니다.



Zedboard에서 강조 표시된 점퍼 설정

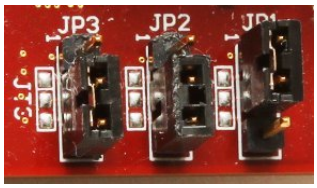
4.1.2 MicroZed

MicroSD 카드에서 Xillinux의 boot를 수행하기 위한 적절한 점퍼 설정은 다음과 같습니다.

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

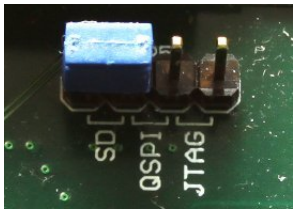
기본 설정이 있는 보드가 주어지면 JP2만 이동하면 됩니다.

올바른 점퍼 설정은 다음과 같습니다.



4.1.3 Zybo

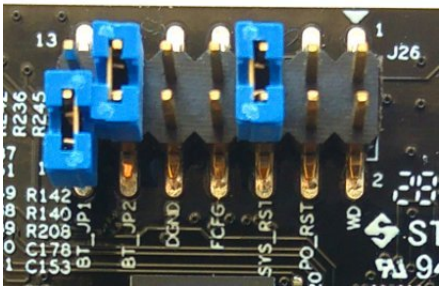
boot mode는 VGA 커넥터 근처에 있는 점퍼에 의해 선택되며, 이 이미지와 같이 “SD”로 표시된 두 개의 핀에 설정해야 합니다.



다른 점퍼는 원하는 작동 모드에 따라 설정됩니다. 예를 들어 전원 공급 장치 점퍼는 외부 5V 소스 또는 UART USB 잭에서 전원을 가져오도록 설정할 수 있습니다. 둘 다 성공적인 Xillinux의 boot에 적합합니다.

4.1.4 Z-Turn Lite

Ethernet 커넥터(J26 레이블) 옆에 있는 점퍼는 boot mode를 결정하며 다음과 같이 설정해야 합니다.



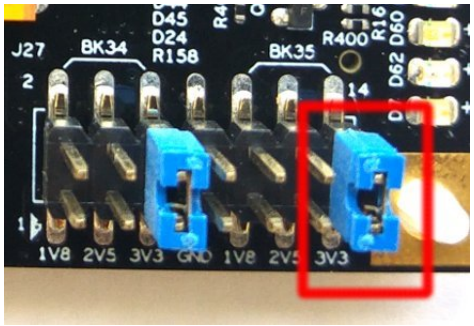
- BT_JP1 점퍼를 배치해서는 안 됩니다(또는 그림과 같이 배치, 즉 핀 중 하나에만 부착됨).
- BT_JP2 점퍼를 배치해야 합니다.
- FCFG 및 PO_RST를 배치해서는 안 됩니다.

boot 작업과 관련이 없는 이 두 점퍼는 몇 가지 중요합니다.

- SYS_RST 점퍼를 사용하면 보드의 K2 버튼을 눌러 Zynq의 processor를 재설정할 수 있습니다.
- WD(Watchdog) 점퍼를 사용하면 processor에서 온보드 watchdog 칩을 활성화할 수 있습니다. Xillinux는 배치 여부에 관계없이 boot를 제대로 수행합니다.

DGND 핀 쌍은 접지에 연결된 두 개의 핀입니다. 여기에 점퍼를 놓아도 아무 효과가 없습니다.

또한 HDMI 비디오 출력이 사용되는 경우(Cape IO 보드를 통해) Zynq 장치의 bank 35는 3.3V 전압 공급으로 구동되어야 합니다. 이것은 MicroSD 카드에 가까운 보드 모서리에서 J27의 BK35 그룹에 있는 3V3 점퍼로 설정됩니다(아래 이미지 참조).



4.2 주변기기 부착

다음 범용 하드웨어를 보드에 부착할 수 있습니다.

- Z-Turn Lite IO Cape board가 있는 Z-Turn Lite: Cape board의 HDMI 커넥터에 대한 컴퓨터 모니터. 또는 HDMI/DVI 어댑터 또는 케이블을 통해 보드의 HDMI 플러그에 연결된 DVI 입력.
- Zedboard / Zybo: 아날로그 VGA 커넥터에 대한 컴퓨터 모니터.
- Zybo: HDMI 입력이 있는 컴퓨터 모니터. 또는 HDMI/DVI 어댑터 또는 케이블을 통해 보드의 HDMI 플러그에 연결된 DVI 입력.
- Zedboard/Zybo/Z-Turn Lite: USB(OTG) 커넥터에 대한 마우스 및 키보드.

  Zybo에는 이를 위한 PC와 유사한 USB 플러그가 있습니다. Zedboard 및 Z-Turn Lite에서 이것은 Zedboard(더 짧은 것)와 함께 제공되는 USB 암 케이블을 통과합니다. 이 케이블은 “kit” 구성으로 구매 시 Z-Turn Lite와 함께 제공됩니다.

  시스템은 이것들이 없을 때 성공적인 boot를 수행하며, 시스템이 실행될 때 키보드와 마우스를 연결하고 분리하는 데 문제가 없습니다. 시스템은 주어진 순간에 연결된 키보드와 마우스를 감지하고 작동합니다.

  Zedboard에서 이 USB 포트가 작동하려면 JP2가 설치되어 있어야 합니다.
- Ethernet 포트는 일반적인 네트워크 작업을 위해 선택 사항입니다. 연결된 네트워크에 DHCP server가 있는 경우 Linux 시스템은 네트워크를 자동으로 구성합니다.
- UART USB 포트는 PC에 연결할 수 있지만 대부분의 경우 Zedboard 및 Zybo에서는 필요하지 않습니다. 일부 boot messages가 그곳으로 전송되고 boot가 완료되면 이 인터페이스에서 shell prompt가 발행됩니다.

  이것은 boot 동안 또는 PC 모니터나 키보드가 없거나 제대로 작동하지 않을 때 오류를 디버깅하는 데 유용합니다.

  Z-Turn Lite의 경우 보드의 3.3V UART 신호와 인터페이스하려면 외부 USB 어댑터가 필요합니다. 이 어댑터는 Z-Turn Lite에 관한 Xillybus의 web page에 설명된 대로 보드에 포함될 수 있습니다.

4.3 보드 전원 켜기

4.3.1 초기 진단

위의 빌드 지침을 따랐을 때 boot 중 오류가 발생하는 경우는 드뭅니다. 일반적인 이유는 다음과 같습니다.

- 잘못된 점퍼 설정(단락 4.1 참조).
- Sandisk에서 제작하지 않은 (Micro)SD 카드 사용. 카드가 제대로 작동하는 것처럼 보이더라도 흩어져 있는 데이터 손상은 쉽게 간과되지만 완전히 다른 이유가 있는 것처럼 보이는 오류가 발생합니다.
- (Micro)SD image를 카드에 불완전하거나 잘못 쓰기
- 보드의 QSPI flash에서 U-boot에 의해 로드된 오래되고 부적절한 환경 변수. 단락 4.3.3를 참조하십시오.
- 지침을 따르지 않음. 일반적으로 시스템을 구축하려는 첫 번째 시도에서 시스템을 조정하려는 시도로 인해 발생합니다.

올바른 UART 설정은 115200 baud, 8 data bits, 1 stop bits이며 flow control은 없습니다. 일부 텍스트는 보드 전원이 켜진 후 4초 이내에 나타나야 합니다. 이를 위한 유일한 요구 사항은 boot.bin 파일이 (Micro)SD 카드의 첫 번째(FAT32) partition에 있다는 것입니다.

보드의 전원을 켠 후 아무 일도 일어나지 않으면:

- 보드 유형과 일치하는 boot partition kit에서 올바른 boot.bin이 복사되었는지 확인합니다. 키트의 파일 이름은 함께 사용해야 하는 보드를 나타냅니다.
- UART 대 컴퓨터 링크가 제대로 작동하는지 확인하십시오. 가능한 QSPI flash 또는 보드와 함께 제공된 SD 카드의 샘플 Linux를 사용하십시오. UART terminal 응용 프로그램용 host인 Linux는 일부 UART/USB 변환기에서 제대로 작동하지 않을 수 있으므로 Windows에서 Tera Term을 시도하는 것이 유일한 옵션일 수 있습니다.

U-boot가 console에서 메시지를 내보내지만 boot 프로세스가 실패하면 4.3.5 단락의 스크립트와 비교하는 것이 도움이 될 수 있습니다. 이 섹션의 나머지 부분에는 무엇이 잘못되었는지 이해하기 위한 관련 정보도 포함될 수 있습니다.

4.3.2 boot 프로세스가 완료되면

boot 프로세스가 끝나면 shell prompt가 UART console에 제공되며 수동으로 로그인할 필요가 없습니다. 그럼에도 불구하고 root user의 암호는 아무것도 설정되어 있지 않으므로 필요할 경우 root로 로그인할 때 암호가 필요하지 않습니다.

중요한:

보드와 함께 제공된 (Micro)SD 카드의 Linux 샘플과 달리 Xillinux의 root file system은 (Micro)SD 카드에 영구적으로 상주하며 시스템이 가동되는 동안 기록됩니다. Linux 시스템은 일반적으로 갑작스러운 정전 후 적절한 복구가 관찰되더라도 다른 PC 컴퓨터와 마찬가지로 시스템을 안정적으로 유지하기 위해 보드의 전원을 끄기 전에 적절하게 종료해야 합니다.

Z-Turn Lite, Zedboard 및 Zybo에 대한 참고 사항:

- shell prompt에서 “startx”를 입력하여 LXDE graphical desktop을 시작합니다. desktop은 초기화하는 데 15-30초 정도 걸립니다. 아무 일도 일어나지 않는 것 같으면 OLED 디스플레이에서 활동 측정기를 모니터링하면 무슨 일이 일어나고 있는지 알 수 있습니다(Zedboard에만 이 OLED 디스플레이가 있음).
- 흰색 배경의 Xillybus 로고 화면 보호기는 logic fabric이 로드되는 순간부터 Linux kernel이 실행될 때까지 화면에 표시됩니다. 또한 운영 체제가 화면을 “blank” 모드로 설정하는 경우(시스템이 유휴 상태일 때 정상 상태) 또는 X-Windows 시스템이 그래픽 모드 조작을 시도할 때 표시됩니다.
- **파란색** 배경의 Xillybus 화면 보호기 또는 화면의 임의의 파란색 줄무늬는 그래픽 인터페이스가 데이터 부족을 겪고 있음을 나타냅니다. 명백한 이유가 알려져 있지 않는 한 이러한 일이 발생할 것으로 예상되지 않으며 보고해야 합니다.

4.3.3 U-boot 환경 변수

Xillinux는 boot 프로세스 중에 xillydemo.bit, kernel image 및 device tree를 로드하기 위해 U-boot를 사용합니다. 이 유틸리티는 다양한 boot 구성을 제공하며 설정을 실험하고 수정할 수 있는 간단한 명령줄 인터페이스가 있습니다.

U-boot가 시작된 직후 UART console에 문자를 입력하면 U-boot의 shell에 도달합니다.

```
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-boot는 항상 QSPI flash에서 저장된 환경 변수를 검색하려고 시도합니다. “bad CRC”라고 표시된 warning은 유효한 데이터가 발견되지 않았음을 나타내므로 U-boot가 하드코딩된 기본 환경으로 되돌아갔습니다. 이것은 (Micro)SD 카드의 Xillinux boot에 대한 오류가 아닙니다. Xillinux의 U-boot에 모든 환경 변수가 올바르게 설정되어 있기 때문입니다.

중요한:

시스템이 (Micro)SD 카드에서 boot를 실행하더라도 환경 변수는 보드 자체의 QSPI flash에서 가져옵니다. QSPI flash가 다른 boot 시나리오와 일치하는 환경 변수를 포함하는 경우, U-boot는 부적절한 변수에 의존하여 boot 프로세스에 실패할 수 있습니다.

1초 동안 아무 키도 누르지 않으면 U-boot는 환경 변수(QSPI flash에서 로드된 변수 또는 하드코딩된 기본 설정)에 따라 계속됩니다. 보다 정확하게는 기본적으로 “run $modeboot”라고 하는 “bootcmd” 변수의 내용을 실행합니다. “modeboot”는 U-boot가 로드된 위치에 따라 U-boot에 의해 동적으로 설정되므로 일반 Xillinux boot에서 “sdboot”라고 합니다. “sdboot” 변수에는 Xillinux의 boot 프로세스를 수행하기 위한 일련의 명령이 포함되어 있습니다.

U-boot의 command-line shell에는 모든 명령과 해당 의미를 나열하는 “help” 명령이 있습니다. 몇 가지 유용한 명령은 다음과 같습니다.

- help *command*– *command*에 대한 도움말 표시
- env print– 모든 환경 변수의 현재 값을 인쇄합니다.
- env set– 특정 환경 변수의 값을 설정합니다.
- env default -a– 모든 환경 변수를 하드코딩된 기본값으로 설정합니다.
- saveenv– 현재 환경 변수를 QSPI flash에 저장합니다(MicroSD/SD 카드가 *아님*).

특히 목록의 마지막 두 명령은 U-boot가 boot 프로세스에 실패할 때 중요합니다. “bad CRC, using default environment”라고 표시된 warning이 U-boot에서 발행 **되지 않은** 경우 저장된 변수에 의존하는 것입니다. 기본 변수(Xillinux에 해당)를 사용하려면 다음으로 이동하십시오.

```
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
```

저장된 환경 변수에 원하는 변경 사항이 있으면 물론 지워집니다.

4.3.4 사용자 지정 Ethernet MAC 주소 설정

기본적으로 Linux는 U-boot가 설정하는 Ethernet MAC 주소를 사용합니다.

MAC 주소를 변경하려면 Network Manager에서 읽는 구성 파일을 추가하면 됩니다. 예를 들어 다음을 /etc/NetworkManager/system-connections/eth0라는 파일에 복사합니다.

```
[connection]
id=eth0
type=ethernet

[ethernet]
cloned-mac-address=00:11:22:33:44:55
mac-address=00:0A:35:00:01:22
```

Network Manager가 구성을 신뢰하도록 이 파일의 permissions를 변경해야 합니다.

```
# cd /etc/NetworkManager/system-connections/
# chmod 0600 eth0
```

“eth0”을 제외하고 이 디렉터리에는 다른 파일이 없어야 합니다. 다른 파일이 있으면 삭제하세요.

Linux를 다시 시작하면 MAC 주소는 00:11:22:33:44:55가 됩니다.

명령줄 유틸리티 “nmcli”를 사용하여 동일한 작업을 수행하는 것도 가능합니다. 그러나 그래픽 데스크탑에서 이 작업을 수행하는 것이 더 쉽습니다. 데스크탑 오른쪽 하단에 있는 Network Manager 아이콘을 클릭하십시오. 이 아이콘은 벽에 연결된 Ethernet 플러그처럼 보입니다. “Edit Connections...”를 선택한 다음 “Wired connection 1”을 선택하고 “Edit”를 클릭합니다. “Cloned MAC address” 옆에 새 MAC 주소를 쓴 다음 “Save”를 클릭합니다. 새로운 MAC 주소는 Linux를 다시 시작한 후에 사용됩니다.

다른 방법은 U-boot의 환경 변수 중 하나를 변경하는 것입니다. U-boot가 항상 QSPI flash에 액세스할 수 있는 것은 아니며 결과적으로 “saveenv” 명령이 항상 지원되는 것은 아니기 때문에 이 방법은 Xillinux의 일부 버전에서는 작동하지 않습니다.

환경 변수는 QSPI flash에 저장되므로 MAC 주소는 하드웨어에 지속적으로 바인딩됩니다. 이는 MAC 주소가 (Micro)SD 카드에 바인딩되는 이전 방법과 다릅니다.

예를 들어 USB UART console을 사용하는 U-boot의 shell에서:

```
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
```

나중에 보드의 전원을 재활용하고 Linux가 boot를 자동으로 수행하도록 한 후:

```
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 주소는 위의 경우 DHCP server에 의해 주어졌습니다)

4.3.5 boot 중 샘플 스크립트

참고로 boot 동안의 일반적인 UART transcript는 아래와 같다. Zedboard에 대한 예가 표시되지만 보드 간의 차이는 미미합니다. boot 프로세스가 실패하면 오류 메시지에 어떤 단계가 잘못되었는지와 그 이유가 표시될 것입니다.

```
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-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
[    0.043389] 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>.
```

그리고 systemd 초기화와 함께 계속됩니다. 30초 이내에 shell prompt가 다음과 같이 나타납니다. 이 최종 출력 조각이 나오기 전에 몇 초의 일시 중지가 있을 수 있습니다.

```
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 첫 번째 boot 직후 수행할 작업

4.4.1 file system 크기 조정

root file system의 image는 가능한 한 빨리 장치에 쓰기 위해 작게 유지됩니다. 반면에 (Micro)SD 카드의 전체 용량을 사용하지 않을 이유가 없습니다.

중요한:

file system의 크기를 조정하려고 시도하는 동안 전체 (Micro)SD 카드의 내용을 지울 상당한 위험이 있습니다. 따라서 가능한 한 빨리 이 작업을 수행하는 것이 좋지만 이러한 사고의 비용은 (Micro)SD 카드 초기화(image 쓰기 및 boot partition 채우기)를 반복하는 것뿐입니다.

시작점은 일반적으로 다음과 같습니다.

```
# 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
```

따라서 root filesystem은 388 MB free와 함께 2.8 GB입니다.

첫 번째 단계는 (Micro)SD 카드를 다시 분할하는 것입니다. shell prompt에서 다음을 입력합니다.

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

그런 다음 다음과 같이 입력합니다(아래 세션 기록 참조).

- d [ENTER]– partition 삭제
- 2 [ENTER]– partition 번호 2 선택
- n [ENTER]– 새 partition 만들기
- [ENTER]를 4번 눌러 기본값을 수락합니다. A primary partition, 2번, 가능한 가장 낮은 sector에서 시작하여 가장 높은 sector에서 끝납니다.
- w [ENTER] – 저장하고 종료합니다.

이 시퀀스 도중에 문제가 발생하면 CTRL-C(또는 q [ENTER])를 눌러 변경 사항을 저장하지 않고 fdisk을 종료하십시오. 마지막 단계까지 (Micro)SD 카드에서 아무 것도 변경되지 않습니다.

일반적인 세션은 다음과 같습니다. sector 번호는 다를 수 있습니다.

```
# 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).
```

시스템에 표시되는 첫 번째 기본 sector가 위의 것과 다른 경우 여기에 표시된 것이 아니라 시스템의 기본값을 선택하십시오.

fdisk의 기본값에서 전환하는 것이 합리적일 수 있는 이 시퀀스의 유일한 위치는 file system을 가능한 최대값보다 작게 만들기 위해 마지막 sector입니다(그러나 이렇게 할 필요는 없습니다).

하단의 warning에서 알 수 있듯이 Linux의 partition table 보기는 업데이트되지 않았습니다. 제안에 따라 다음을 입력합니다.

```
# partprobe
```

이렇게 하면 console에 출력이 생성되지 않습니다. partition 2의 변경 사항을 kernel에 알릴 수 없다고 불평하는 경우 partprobe이 이를 찾을 수 없기 때문일 가능성이 큽니다. 즉, root partition은 partition table이 있어야 한다고 말하는 곳이 아닙니다. 아마도 fdisk에 문제가 발생하여 수정해야 합니다. 그렇지 않으면 Linux가 boot를 다시 실행할 수 없습니다.

partprobe이 조용했다면 partition table은 괜찮지만 file system은 아직 크기가 조정되지 않았습니다. 크기를 조정할 수 있는 공간만 주어졌습니다. 따라서 shell prompt에서 다음을 입력하십시오.

다음 응답이 예상되는

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

:

```
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 count는 partition의 크기에 따라 달라지므로 다를 수 있습니다.

유틸리티에서 알 수 있듯이 크기 조정은 활발하게 사용되는 file system에서 발생합니다. 중간에 전원이 손실되지 않는 한 안전합니다.

결과는 즉시 적용됩니다. 재부팅할 필요가 없습니다.

8 GB (Micro)SD 카드를 사용하는 일반적인 세션:

```
# 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" 유틸리티에서 제공하는 크기는 1 GiB = 2^{30} 바이트로, 10^9 바이트의 Gigabyte보다 7.1%가 더 큽니다. 이것이 8 GB 카드가 위에서 7.1 GiB로 나타나는 이유입니다.

4.4.2 원격 SSH 액세스 허용

root password는 기본적으로 없음이므로 모든 사용자가 암호 없이 root로 로그인할 수 있습니다. 결과적으로 ssh는 root 로그인을 거부합니다.

이를 수정하려면 shell prompt에서 다음 명령으로 root password를 설정하십시오.

```
# passwd
```

4.4.3 Compilation 로케일 정의(필요한 경우)

특정 상황에서 응용 프로그램 소프트웨어는 locale settings에 따라 문자를 표시하는 방법에 대한 지식이 필요합니다. 가장 일반적인 이유는 ssh를 보드에 연결할 때 ssh가 shell session을 설정할 때 client의 locale settings를 server의 환경으로 복사하기 때문입니다.

이것은 다음과 같은 warning messages로 이어질 수 있습니다.

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

이러한 오류 메시지가 나타나면 어떤 locale이 누락되었는지 알 수 있습니다. 오류가 발생하기 전에 이 문제를 해결하려면 어떤 locales가 필요한지 확인하십시오.

```
# 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=
```

사용 가능한 locales와 비교:

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

이 예에서는 어떤 locale이 누락되었는지 매우 분명하므로 추가해 보겠습니다.

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

필요한 locale은 ssh가 연결된 컴퓨터에 따라 다릅니다. 세계 여러 곳의 사용자는 원활한 ssh 세션을 달성하기 위해 보드에 다른 locales를 설치해야 합니다. UART 포트의 shell은 기본적으로 포함된 POSIX locale을 기반으로 합니다.

en_US.UTF-8은 유비쿼터스이기 때문에 이미 배포판에 설치되어 있습니다(위의 예제 세션이 반대를 보여주더라도).

4.5 desktop 사용

Xillinux desktop(Z-Turn Lite, Zedboard 및 Zybo)는 모든 Lubuntu desktop와 같습니다. (Micro)SD 카드의 상대적으로 낮은 데이터 대역폭으로 인해 응용 프로그램이 다소 느리게 로드될 수 있지만 desktop 자체는 상당히 응답합니다.

Ubuntu 배포판과 마찬가지로 추가 패키지를 “apt-get”와 함께 설치할 수 있습니다.

전체 Ubuntu 운영 체제를 apt로 업그레이드하는 것은 불가능하며 실패하여 시스템에서 boot를 다시 수행할 가능성이 없습니다.

4.6 종료/재부팅

시스템 전원을 끄려면 바탕 화면의 오른쪽 하단 아이콘(사용 가능한 경우)을 선택하고 “Shutdown”을 클릭하거나 적절한 다른 옵션을 선택합니다. “Lock Screen”은 아무것도 하지 않습니다.

또는 shell prompt에 다음을 입력합니다.

```
# halt
```

“System Halted”라는 텍스트 메시지가 UART console에 나타나면(그리고 화면에 있는 경우) 보드의 전원을 끄는 것이 안전합니다.

bitstream을 FPGA(PL) 부품으로 다시 로드하는 것을 포함하는 reboot의 경우 바탕 화면 메뉴에서 reboot 옵션을 선택하거나 다음을 입력합니다.

```
# reboot
```

사운드 칩과 같은 외부 하드웨어 구성 요소가 반드시 재설정되는 것은 아닙니다.

4.7 여기에서 무엇을해야합니까

Zynq 보드는 이제 모든 용도로 Linux를 실행하는 컴퓨터가 되었습니다. Xillybus IP core를 통해 logic fabric와 상호 작용하는 기본 단계는 Getting started with Xillybus on a Linux host에서 찾을 수 있습니다. Xillybus용 driver는 이미 Xillinux 배포판에 설치되어 있으므로 가이드에서 설치를 다루는 부분은 건너뛸 수 있습니다.

5.1 단락은 응용 프로그램별 logic을 Linux 운영 체제와 통합하는 것을 나타냅니다.

Xillinux에는 gcc compiler 및 GNU make이 포함되어 있으므로 보드의 processors에서 직접 일반 컴퓨터 프로그램의 compilation을 실행할 수 있습니다. apt-get와 함께 배포판에 추가 패키지를 추가할 수도 있습니다.

5

수정

5.1 맞춤형 logic와 통합

Xillinux 배포는 응용 프로그램 logic와 쉽게 통합되도록 설정됩니다. 데이터 소스와 데이터 소비자를 연결하기 위한 프런트 엔드는 xillydemo.v 또는 xillydemo.vhd 파일입니다(선호하는 언어에 따라 다름). boot partition kit의 다른 모든 HDL 파일은 Linux host와 logic fabric 간의 데이터 전송으로 Xillybus IP core를 사용하기 위해 무시할 수 있습니다.

사용자 지정 logic designs가 있는 추가 HDL 파일을 3.3 단락에 제시된 프로젝트에 추가한 다음 처음에 수행한 것과 동일한 방식으로 다시 빌드할 수 있습니다. 업데이트된 logic로 시스템의 boot를 실행하려면 새 xillydemo.bit을 (Micro)SD 카드의 boot partition에 복사하여 기존 카드를 덮어씁니다. 단락 3.5에 표시된 대로 xillydemo.bit을 boot partition로 복사하기 위해 Zynq 보드 자체를 사용할 수 있습니다.

초기 배포 배포의 다른 단계를 반복할 필요가 없으므로 logic의 개발 주기는 상당히 빠르고 간단합니다.

JTAG을 통한 PL 부품 프로그래밍은 지원되지 않습니다.

Xillybus IP core를 사용자 지정 application logic에 연결할 때 FIFOs를 통해서만 Xillybus IP core와 상호 작용하고 logic에서 FIFO의 동작을 모방하려고 시도하지 않는 것이 좋습니다. 최소한 첫 번째 단계에서는 아닙니다.

이에 대한 예외는 Xillybus를 block RAM 또는 registers와 연결할 때이며, 이 경우 xillydemo 모듈에 표시된 방법을 따라야 합니다.

xillydemo 모듈에서 FIFOs는 host에서 도착한 데이터의 loopback을 수행하고 다시 host로 돌아오는 데 사용됩니다. FIFOs의 양쪽 측면은 Xillybus IP core에 연결되어 있어 core가 자체 데이터 소스 및 데이터 소비자 역할을 합니다.

더 유용한 시나리오에서는 FIFO의 한쪽 끝만 Xillybus IP core에 연결되고 다른 쪽 끝은 애플리케이션 데이터 소스 또는 데이터 소비자에 연결됩니다.

xillydemo 모듈에 사용된 FIFOs는 양쪽 모두 Xillybus의 기본 clock에 의해 구동되기 때문에 양쪽에 대해 하나의 공통 clock만 허용합니다. 실제 애플리케이션에서는 읽기 및 쓰기를 위한 별도의 clocks가 있는 FIFOs로 교체하는 것이 바람직할 수 있으므로 데이터 소스와 데이터 소비자가 bus clock이 아닌 clock에 의해 구동될 수 있습니다. 이렇게 함으로써 FIFOs는 중재자 역할을 할 뿐만 아니라 적절한 clock domain crossing의 역할도 합니다.

Xillybus IP core는 FPGA에서 host까지의 streams에 대해 *일반* FIFO(First Word Fall Through와 반대)를 예상합니다.

다음 문서는 맞춤형 logic 통합과 관련이 있습니다.

- logic design용 API: Xillybus FPGA designer's guide
- Linux host의 기본 개념: Getting started with Xillybus on a Linux host
- 프로그래밍 애플리케이션: Xillybus host application programming guide for Linux
- 맞춤형 Xillybus IP core 요청: The guide to defining a custom Xillybus IP core

5.2 다른 보드 사용

Z-Turn Lite, Zedboard, MicroZed 또는 Zybo 이외의 보드에서 Xillinux를 실행하기 전에 특정 수정이 필요할 수 있습니다.

그러나 절차가 어렵기 때문에 Xillinux를 다른 하드웨어에 적용하는 것은 권장하지 않습니다. 경험에 따르면 Xillinux를 적용하는 목적이 Xillybus IP core를 사용하는 것이 아닌 경우 처음부터 시작하는 것이 더 쉽습니다.

주의해야 할 문제의 일부 목록입니다.

- 구입한 보드에는 XML 파일이 참조로 있어야 합니다(ps7_system_prj.xml로 사용하기 위해). 이 파일에는 MIO 핀의 사실상 사용과 DDR 핀의 전기적 매개변수를 포함하여 processor의 설정이 포함되어 있습니다. 권장되는 방법은 참조 파일을 최소한 시작점으로 채택하는 것입니다.
- XML 파일이 참조로 채택되면 참조 파일의 내용에 관계없이 FPGA CLK1(FCLK_CLK1)를 100 MHz로 설정해야 합니다.
- 수동으로 변경하는 경우 processor core의 MIO 할당에 주의를 기울여야 합니다. ARM core에는 고정 배치로 칩의 물리적 핀으로 라우팅되는 54개의 I/O pins가 있습니다. ARM core는 프로젝트의 block design에서 이러한 핀에 특정 역할을 할당하도록 구성되어 있습니다(예: USB 인터페이스, Ethernet 등). 이는 보드에서 이러한 핀이 연결된 것과 일치해야 합니다.

- processor의 구성이 변경되면(즉, XML file에서) boot.bin은 새 XML 파일에서 파생된 FSBL (First Stage Boot Loader)와 U-boot binary를 기반으로 다시 빌드해야 합니다. Vivado의 block design 도구에서 변경된 사항은 FSBL의 일부인 초기화 루틴을 통해 적용됩니다. 이 루틴은 Vivado에서 이루어진 설정을 반영하고 SDK로 내보낸 값으로 ARM processor의 registers에 기록합니다. Vivado 프로젝트의 processor 매개변수가 정확하지 않을 수 있으므로 FSBL은 번들에서 사용 가능한 XPS 프로젝트를 기반으로 생성되어야 합니다. U-boot에 대한 소스를 설정하려면 /usr/src/xillinux/uboot-patches/의 README 파일을 참조하십시오.
- 또는 경우에 따라 “poke” 기능 덕분에 registers 설정의 변경 사항을 정확히 찾아 boot.bin을 재구축하는 것을 피할 수 있습니다. 단락 5.6를 참조하십시오.
- 새 설정을 반영하기 위해 devicetree.dtb를 변경해야 할 수도 있습니다. 기존 DTB(DTS 형식)의 소스는 Linux kernel의 소스에서 찾을 수 있습니다(단락 6.2 참조).
- VGA/DVI 출력(해당되는 경우)은 의도한 보드와 일치해야 합니다. 이것은 src/ 하위 디렉토리에서 xillybus.v 파일을 편집하여 수행됩니다. “system” 모듈에서 도착하는 신호의 너비는 8비트이며 xillybus.v에서는 4비트로 잘립니다. 따라서 이러한 신호를 VGA/DVI용 인코더 칩에 연결하는 것은 매우 쉽습니다.

5.3 시스템에서 clocks의 주파수 변경

ARM processor의 core는 일반적으로 FCLK_CLKn라고 하는 logic fabric에서 사용할 4개의 clocks를 제공합니다. 주파수는 U-boot가 로드되기 전에 FSBL (First Stage Boot Loader)에 의해 설정된다는 점에 유의하는 것이 중요합니다.

따라서 clocks의 주파수가 Vivado에 설정되어 있더라도 이러한 주파수는 SDK의 compiled인 bare-metal 응용 프로그램에 의한 timing constraints 전파 및 초기화에만 유효합니다.

하드웨어 응용 프로그램에 다른 주파수가 필요한 경우 다음과 같은 일련의 작업이 제안됩니다.

- Vivado에서 clock의 주파수를 업데이트합니다.
- 넷리스트 재구축(.ncf 파일에서 timing constraints 업데이트에 필요)
- 프로젝트를 SDK로 내보내고 이를 기반으로 FSBL application project를 만듭니다.
- Vivado의 보고서에서 원하는 설정에 필요한 registers의 설정을 알아보십시오.
- 5.6 단락에 설명된 대로 “poke” 기능을 사용하여 필요에 따라 조정합니다.

각 단계를 수행하는 방법에 대한 자세한 내용은 Xilinx 가이드를 참조하십시오(마지막 단계 제외).

5.4 PL logic용 GPIO I/O 핀 인수

5.4.1 Z-Turn Lite

Z-Turn Lite 보드 자체는 연구 목적으로 I/O 핀에 액세스할 수 있는 편리한 방법을 제공하지 않지만 Z-Turn Lite IO Cape board에 연결하면 HDMI 인터페이스 외에도 표준 커넥터를 통해 68 I/O pins와 푸시 버튼이 노출됩니다.

이 68핀은 모두 표준 플랫 리본 케이블을 연결할 수 있는 2개의 40핀 커넥터 J3 및 J8에 연결되어 있습니다. IO Cape board에는 J3 및 J8와 핀을 공유하는 몇 가지 추가 커넥터가 있습니다. 모든 추가 커넥터의 핀은 J3 및 J8에서 사용할 수 있으므로 이러한 핀에 대한 Xillydemo의 top-level module 포트는 pin placement constraints가 해당 커넥터로 라우팅하는 J3 및 J8라는 벡터입니다.

J3 및 J8의 Verilog / VHDL 포트에 있는 벡터는 커넥터의 핀 번호에서 3을 뺀 값에 해당합니다. Verilog / VHDL의 신호 J3[0]은 물리적 핀 J3/3로 이동합니다. J3[1]은 J3/4로 이동하고 J3[33]은 J3/36로 이동합니다. J8도 마찬가지입니다.

단순화를 위해 J8에 속한 모든 핀은 xillydemo.v 및 xillydemo.vhd에서 processor의 GPIO 핀에 연결되며 processor에서 실행되는 소프트웨어에서 직접 제어할 수 있습니다. J3의 모든 핀은 xillydemo.v 및 xillydemo.vhd에 의해 로우로 구동되며 관련 xillydemo 모듈 파일을 수정하여 애플리케이션 logic에서 쉽게 사용할 수 있습니다.

GPIO에 대한 핀 분할과 application logic에 의해 구동되는 핀도 xillydemo 모듈에서 배선을 변경하여 쉽게 변경할 수 있습니다. GPIO로 사용되는 핀 수가 변경되면 xillybus 모듈의 gpio_width 인스턴스화 매개변수(일반)도 그에 따라 변경되어야 합니다. 현재 35개로 J8 커넥터로 가는 34개의 I/O 핀과 Cape Board의 푸시 버튼용 GPIO 1개를 포함합니다.

그리고 앞에서 언급했듯이 보드에는 여전히 사용할 수 있는 J3 및 J8와 핀을 공유하는 추가 커넥터가 있습니다. 그러나 이를 위해서는 Cape Board의 회로도에서 어느 핀이 어디로 가는지 찾아야 합니다.

이 핀 공유의 한 가지 부작용은 대체 커넥터에 의해 I^2C로 사용되는 일부 핀이 Cape board에 pull-ups가 있다는 것입니다: J3[19], J3[18], J8[28] 및 J8[31](Verilog / VHDL 벡터 신호 표기법 사용). 보드에 저항이 있는 pull-ups이므로 J3 및 J8 커넥터에 이 핀을 사용하는 경우에도 유효합니다.

HDMI 커넥터는 독립적이며 J3, J8 또는 다른 커넥터와 핀을 공유하지 않습니다.

5.4.2 Zedboard 및 Zybo

Zedboard 및 Zybo 보드에서 많은 물리적 I/O 핀이 ARM processor의 GPIO 포트(PS)에 연결되어 Linux에서 직접 이러한 핀을 제어하고 모니터링할 수 있습니다. 그러나 이러한 물리적 핀을 대신 FPGA logic에 연결하는 것이 좋습니다(즉, PL).

I/O에 Zynq의 PL 핀을 사용하는 기술은 모든 Xilinx FPGA와 완전히 동일합니다. 신호는 최상위 모듈(xillydemo.v 또는 xillydemo.vhd)에서 입력, 출력 또는 inouts로 노출됩니다. 이러한 신호에 대한 물리적 핀 할당은 xillydemo.xdc에서 발생합니다.

핀은 GPIO 신호로 사용되기 때문에 processor에서 빼내어 PL 부품에 제공됩니다. 예를 들어 XDC 파일에서 my_output이 핀 U5에 나타나도록 하려면

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

를

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

로 바꿀 수 있습니다.

그러나 이 교체를 수행하면 PS_GPIO[55]에 핀 할당이 부족합니다. Xilinx의 도구가 implementation 동안 이 포트를 자동으로 배치할 가능성이 있더라도 축출된 PS_GPIO에 I/O 핀을 할당하는 것이 좋습니다. 대안은 아래에 설명된 대로 신호를 제거하는 것입니다.

따라서 이러한 제거된 PS_GPIO 신호에 대한 두 가지 솔루션이 있습니다.

- 쉬운 방법: 장치에서 사용하지 않는 핀을 찾아 제거된 PS_GPIO 신호에 이 핀을 할당합니다. 매우 깨끗한 솔루션은 아니지만(GPIO 핀은 보드의 모든 것에 연결됨) GPIOs가 기본적으로 입력이기 때문에 사실상 무해합니다. GPIO가 소프트웨어에 의해 우발적으로 구동되지 않는 한 이 핀의 전기적 상태는 유지됩니다(가능성은 낮음). 예를 들어, Zedboard에서 FMC 커넥터는 종종 사용하지 않는 많은 핀을 제공합니다.
- 더 어려운 방법: PS_GPIO 핀 수 줄이기. 이것은 빈 핀이 많지 않은 Zybo에서 필요할 수 있습니다.

다음에서는 두 번째 솔루션에 대해 설명합니다. 예를 들어 PS_GPIO[55:48]이 PL의 신호로 핀을 교체하기 위해 XDC 파일에서 제거되었다고 가정해 보겠습니다. 낮은 PS_GPIO 인덱스의 핀이 필요한 경우 제거된 PS_GPIO 신호가 인덱스가 가장 높은 핀의 핀을 인수해야 하며 후자는 제거됩니다. 특정 범위의 PS_GPIO 인덱스를 제거할 가능성은 없으며 최대 인덱스만 줄입니다.

PS_GPIO의 너비는 XDC 파일에 핀 할당이 있는 것을 반영하기 위해 xillydemo.v/vhd에서 줄여야 합니다.

그러나 이것으로 충분하지 않습니다. 이 상태에서 프로젝트를 빌드하려고 하면 이 핀에 대해 critical warnings가 발행됩니다(high-Z 및 GND를 사용하여 다중 구동된다고 주장할 수 있음).

이 문제를 해결하려면 다음 부분에서 vivado-essentials/system.v를 편집하십시오.

```
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
```

인덱스 범위(즉, `i<56` 부분)를 사용된 GPIOs의 수(예시에서는 48)로 줄이십시오.

Vivado의 개정판에 따라 신호의 너비를 조정해야 할 수도 있습니다.

block design에서 GPIO 너비를 수정해야 하는 경우: Vivado의 기본 창에서 왼쪽 열의 “Open Block Design”을 클릭합니다. processor block(processing_system_7_0, ZYNQ 표시 있음)를 마우스 오른쪽 버튼으로 클릭하고 “Customize block”을 선택합니다. 왼쪽 열에서 “MIO Configuration”을 선택하고 “I/O Peripherals” 계층을 확장한 다음 GPIO 계층(하단)을 확장합니다. EMIO GPIO (Width) 매개변수는 현재 GPIO 핀 수인 56개입니다. 원하는 숫자(이 예에서는 48)로 줄이십시오.

5.5 7020 MicroZed로 작업

MicroZed에 사용 가능한 boot partition kit는 기본적으로 7010 MicroZed 보드용입니다. 그러나 Vivado 프로젝트를 생성하는 데 사용된 xillydemo-vivado.tcl 파일(즉, 선택한 언어에 따라 번들의 verilog/xillydemo-vivado.tcl 또는 vhdl/xillydemo-vivado.tcl)을 약간 변경한 후 Vivado를 사용하여 7020 MicroZed로 작업할 수 있습니다.

키트의 압축을 푼 직후(Vivado에서 사용하기 전에) 파일을 편집하여 다음 줄을 변경해야 합니다.

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

(라인 11 주변)

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

나머지 빌드 프로세스는 완전히 동일합니다.

5.6 hardware registers의 boot 이전 조작(“poke”)

boot.bin 파일을 재구축하지 않고 ARM processor의 하드웨어 설정을 약간 변경하는 것이 종종 바람직합니다(5.3 단락은 일반적인 재구축 순서를 설명함).

예를 들어, processor의 MIO/EMIO 구성이 약간 변경되면 registers의 설정이 약간 변경되며, 이는 시스템 설정을 소프트웨어 도구로 내보낼 때 생성되는 보고서의 차이점을 찾아 매우 쉽게 추론할 수 있습니다.

processor의 hardware registers는 Xilinx의 Zynq-7000 AP SoC Technical Reference Manual(TRM 또는 ug585라고도 함)에 문서화되어 있습니다.

registers를 조작하기 위해 항목이 kernel의 device tree에 추가됩니다(일반적으로 Linux kernel 소스에 제공된 각 DTS 파일을 편집하여 6.2 단락 참조).

다음 예와 같은 항목은 device tree 계층 구조의 아무 곳에나 추가됩니다(바람직하게는 “chosen” 항목 뒤에).

```
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
                   >;
};
```

원하는 register 읽기 및 쓰기 시퀀스를 설정하려면 “sequence” 부분을 변경해야 합니다. 각 작업은 “sequence” 요소 배열의 세 값으로 정의됩니다. 트리플렛(및 그에 따른 작업)의 수에는 제한이 없습니다. tabs 및 행당 3개의 값을 사용하는 위의 “sequence” 항목 형식은 구문상 의미가 없습니다. 중요한 것은 각 트리플렛이 다음과 같이 작업을 나타내는 것입니다.

- 첫 번째 요소: 읽기 또는 쓰기. 값 0은 읽기를 의미하고 그렇지 않으면 쓰기를 의미합니다.

- 두 번째 요소: 주소. 32비트로 정렬되어야 합니다(주소의 2 LSBs는 0이어야 함).
- 세 번째 요소: 쓸 값입니다. 읽기 작업에서 무시됩니다.

작업은 device tree 항목에 나열된 순서대로 수행되며 각 작업 간에 예측할 수 없는 지연이 발생합니다.

실질적인 의미가 없는 위의 예에서 processor의 ttc2 (Triple Timer Counter 2)의 registers가 조작됩니다. 처음 세 개의 작업은 데모를 위해 registers를 읽습니다. 그런 다음 카운터가 잠시 활성화되고 카운터 값이 두 번 읽혀 변경되고 있음을 표시한 다음 카운터가 비활성화됩니다. 그 후 카운터 값을 다시 두 번 읽어서 중지되었음을 나타냅니다.

이러한 작업의 결과는 serial console (UART)에서 사용할 수 있는 kernel의 message log 및/또는 shell prompt에서 dmesg 명령으로 찾을 수 있습니다.

```
[    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
```

0xf8002018에서 읽은 값은 물론 실행 중인 카운터에서 읽은 값으로 다양합니다.

register 수정은 device driver가 로드되기 전에 kernel의 boot 프로세스 초기에 발생합니다. 그러나 U-boot는 Linux가 boot 프로세스를 시작하기 전에 ARM processor의 하드웨어 주변 장치 중 일부를 설정하므로 이미 활성화되어 있습니다. 또한 ARM processor의 기본 기능(예: clocks 및 interrupts)과 관련된 registers를 수정하면 processor 자체의 적절한 기능이 중단될 수 있습니다. 이것은 “poke”가 실행될 때 kernel이 interrupts를 비활성화한 경우에도 발생할 수 있습니다.

kernel의 메모리 관리 및/또는 하드웨어 자체에 의해 허용되지 않은 주소에 액세스하려고 하면 kernel Oops, kernel panic이 발생하고 완전히 정지될 수 있습니다. 후자의 두 가지는 boot 오류로 이어집니다. “imprecise external abort (0x406)”에 근거한 kernel panic은 아마도 하드웨어 방식의 잘못된 주소에 액세스하려는 시도 때문일 것입니다.

또한 초기 kernel console 메시지는 생성 단계에서 내부 memory buffer에 저장되고 이후 단계(serial port가 설정될 때)에서만 console에 기록되기 때문에 조기 정지로 인해 출력이 없을 수 있습니다. console에 전혀 – U-boot의 “done, booting the kernel” 메시지 뒤에 아무것도 나타나지 않습니다.

따라서 console에 kernel messages가 나타나지 않는다고 해서 반드시 kernel이 시작되지 않았다는 의미는 아닙니다. 메모리에 저장된 kernel messages가 console에 쓰기 전에 정지된 결과일 수 있습니다. 그 이유는 register의 불법 수정일 수 있습니다.

“poke” 기능은 특히 Xillinux-2.0의 kernel을 패치하여 추가되었으며 메인 라인 Linux kernels의 일부가 아닙니다.

6

Linux 노트

6.1 일반적인

이 섹션에는 Xillinux와 관련된 Linux 관련 항목이 포함되어 있습니다.

Xillybus 및 Linux에 대한 더 넓은 관점은 다음 문서에서 찾을 수 있습니다.

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

6.2 Linux kernel의 Compilation

크기 때문에 전체 Linux kernel은 Xillinux 배포판에 포함되어 있지 않지만 다음을 사용하여 Github에서 다운로드할 수 있습니다.

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

Xillinux-2.0와 함께 사용되는 kernel의 정확한 복제는 “xillinux-2.0a” 태그를 확인하십시오. 다음과 같은 형식으로 원하는 cross compiler의 kernel 빌드 환경에 알립니다.

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

그런 다음 Xillinux-2.0와 함께 제공되는 kernel의 compilation에 사용되는 .config 파일을 가져옵니다.

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

그런 다음 kernel의 compilation을 실행합니다.

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

Device Tree Blobs를 빌드하기 위해 위에서 말한 대로 cross compiler와 .config 파일을 설정한 후 다음 명령을 사용할 수 있습니다.

```
$ 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 파일은 명령 응답에서 제안한 대로 arch/arm/boot/dts/에서 찾을 수 있습니다.

6.3 kernel modules의 Compilation

Xillinux 배포판은 실행 중인 kernel의 compilation headers와 함께 제공됩니다. 이것은 kernel 자체의 compilation을 수행하기에 충분하지 않지만 플랫폼에서 직접 kernel modules의 compilation을 허용합니다.

트리 외부 kernel module의 compilation에 대한 표준 방법은 environment variable을 설정한 후 특정 모듈의 compilation을 수행하도록 지시하는 kernel의 자체 빌드 환경을 호출하는 Makefile을 사용하는 것입니다.

단일 소스 파일 example.c로 구성된 kernel module의 *기본* compilation(Zynq processor 자체에서 수행)에 대한 최소 Makefile은 다음과 같습니다.

```
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
```

모듈 이름 “example”은 “obj-m” 라인에서만 언급됩니다. 이것은 Makefile와 다른 Makefile의 유일한 차이점입니다.

이 Makefile을 사용하면 일반적으로 다음과 같은 compilation 세션이 생성됩니다(보드 자체에서 실행, cross compilation 아님).

```
# 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 사운드 지원

6.4.1 일반적인

Zedboard 및 Zybo 보드는 각각 Analog Devices의 ADAU1761 및 SSM2603 칩셋 덕분에 사운드 녹음 및 재생을 지원합니다. 이들은 Zynq 장치의 logic fabric (PL) 핀에만 연결됩니다.

명백한 기능을 제외하고, 사운드 지원 패키지는 또한 Xillybus IP core를 사용하여 데이터를 전송하고 SMBus/I^2C를 통해 칩을 프로그래밍하는 방법도 보여줍니다.

Xillinux는 전용 Xillybus streams를 오늘날 Linux의 가장 일반적인 사운드 툴킷인 Pulseaudio와 인터페이스하여 기본적으로 사운드를 지원합니다. 결과적으로 사운드 카드가 필요한 거의 모든 응용 프로그램은 보드의 사운드 칩을 시스템의 기본 입력 및 출력으로 적절하게 사용합니다.

Pulseaudio daemon을 끄고 Xillybus streams(/dev/xillybus_audio)와 직접 작업하는 것도 가능합니다. 이것은 다른 애플리케이션의 기본 기능을 상실하는 대가로 device file을 여는 단순한 프로그래밍 인터페이스를 제공합니다.

Linux 사운드 카드에 대한 일반적인 접근 방식과 달리 사운드 인터페이스를 위한 전용 kernel driver(예: ALSA)가 없습니다. Xillybus host driver가 어쨌든 데이터 전송을 처리하기 때문입니다. Pulseaudio에는 “padsp” 유틸리티를 사용하여 이 인터페이스를 가장할 수 있는 기능이 있기 때문에 /dev/dsp와 함께 작동할 것으로 예상되는 프로그램이 아니더라도 이것은 의미가 없습니다.

6.4.2 사용 세부 정보

기본적으로 사운드는 헤드폰 잭(검정색)으로 재생됩니다. Zedboard에서 동일한 출력이

Line Out(녹색)에도 나타납니다. 녹음에는 마이크 입력(분홍색)만 사용되지만 다음 설명과 같이 변경할 수 있습니다.

6.4.3 관련 boot scripts

Zedboard 및 Zybo에서 사운드 설정과 관련된 두 가지 작업이 있습니다. 오디오 칩 프로그래밍 및 Pulseaudio daemon 실행.

첫 번째 작업의 경우 /etc/systemd/system/에 두 개의 systemd unit files가 있습니다.

- xillinux_sound.service: 시작 시 /usr/local/bin/xillinux-sound를 시작합니다.
- xillinux_sound.path: systemd가 /dev/xillybus_smb가 나타날 때까지 기다리라고 지시하고, 그런 일이 발생하면 방금 언급한 service를 시작합니다. 이것은 udev를 기반으로 하는 것이 아니라 /dev를 모니터링하는 inotify를 기반으로 한다는 점에 유의하십시오.

/usr/local/bin/xillinux-sound는 실행되는 보드를 식별하고 해당하는 경우 script /usr/local/bin/zybo_sound_setup.pl 또는 /usr/local/bin/zedboard_sound_setup.pl을 실행합니다.

Pulseaudio daemon은 /etc/systemd/user/에 있는 사용자 서비스 단위 파일 xillinux_pulseaudio.service 덕분에 사용자가 시스템에서 첫 번째 로그인 세션을 생성할 때 시작됩니다.

Xillinux는 console 화면뿐만 아니라 직렬 console에서도 root 사용자를 자동 로그인하므로 시스템의 boot가 완료된 직후 Pulseaudio가 시작됩니다.

Pulseaudio를 root로 실행하는 것은 다중 사용자 컴퓨터에서 권장되지 않지만 Xillinux는 기본 사용자로 root와 함께 실행되기 때문에 이것이 가장 문제가 적은 옵션입니다.

예를 들어 /dev/xillybus_audio에 직접 액세스해야 하는 경우 서비스를 비활성화해야 합니다.

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

zybo_sound_setup.pl와 zedboard_sound_setup.pl은 오디오 칩의 registers가 제대로 작동하도록 설정한 Perl scripts입니다. Perl에 익숙하지 않은 프로그래머에게도 매우 간단합니다. script는 /dev/xillybus_smbus device file을 사용하여 칩의 I^2C bus에서 트랜잭션을 시작합니다.

zedboard_sound_setup.pl을 편집하여 오디오 칩의 다른 설정을 얻을 수 있습니다. 특히, Line In 입력이 원하는 녹음 소스인 경우

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

라인을 다음으로 교체해야 합니다.

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

유사한 방식으로 zybo_sound_setup.pl을 편집하여

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

를

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

로 교체하여 Line In을 녹음에 사용할 수 있습니다.

6.4.4 /dev/xillybus_audio에 직접 액세스

/dev/xillybus_audio는 재생을 위해 직접 쓰거나 녹음을 위해 읽을 수 있습니다. sample format은 오디오 샘플당 32비트이며 little Endian 형식의 16비트 signed integers 2개로 나뉩니다. 가장 중요한 단어는 왼쪽 채널에 해당합니다.

샘플링 속도는 48000 Hz로 고정됩니다.

이 샘플링 속도의 Windows WAV 파일은 /dev/xillybus_audio에 직접 기록된 경우 올바르게 재생될 가능성이 높습니다(header는 약 1 ms에서도 재생됨). 예:

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

응답이

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

이면 다른 process가 쓰기를 위해 device file을 열고 있을 가능성이 있습니다. 아마도 Pulseaudio daemon일 것입니다. device file을 한 프로세스에서는 읽고 다른 프로세스에서는 쓰기 위해 열어도 문제가 없습니다.

6.4.5 Pulseaudio 세부 정보

Pulseaudio는 두 개의 전용 Pulseaudio 모듈, module-file-sink 및 module-file-source를 통해 /dev/xillybus_audio device file와 상호 작용합니다. 해당 소스는 Xillinux의 file system(/usr/src/xillinux/pulseaudio/)에서 찾을 수 있습니다.

이 모듈은 UNIX pipes를 데이터 sinks 및 소스(module-pipe-sink 및 module-pipe-source)로 사용하기 위해 표준 Pulseaudio 모듈을 약간 수정한 것입니다.

/etc/pulse/default.pa의 다음 두 줄 덕분에 Pulseaudio가 시작될 때 모듈이 자동으로 로드됩니다.

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

이 모듈은 다른 대안이 없기 때문에 자동으로 시스템의 사운드 인터페이스로 선택됩니다.

6.5 OLED 유틸리티(Zedboard만 해당)

기본적으로 Xillinux는 boot 동안 활동 측정기 유틸리티를 시작하여 대략적인 CPU 사용 백분율과 SD flash disk의 I/O 속도 표시를 표시합니다.

CPU 백분율은 /proc/stat을 기반으로 하며, 여기서 유휴 상태로 소비되지 않은 모든 시간은 사용된 CPU 시간으로 간주됩니다.

SDIO 트래픽의 추정은 인터럽트가 각 driver로 전송되는 속도를 기반으로 합니다. 이 리소스의 전체 활용도에 대한 알려진 수치는 없습니다. 오히려 유틸리티는 이전 측정에 따라 최대로 나타나는 인터럽트 비율에 대해 100%를 표시합니다.

보드의 OLED에 그래픽 출력을 표시하기 위해 bitmap은 Digilent의 driver에 의해 생성된 /dev/zed_oled로 전송됩니다. 이 driver는 bit-banging 메커니즘을 사용하여 OLED 장치에 SPI 데이터를 전송하여 소프트웨어의 clock와 데이터를 토글한다는 점을 언급할 가치가 있습니다. 따라서 초당 여러 번 512바이트를 보내는 약간의 CPU 소비가 있지만 전체 시스템 성능에 미치는 영향은 미미합니다.

이 유틸리티의 매개변수를 변경하려면 다음 명령으로 요약되는 /usr/local/bin/start_zedboard_oled를 편집하십시오.

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

애플리케이션 zedboard_oled는 세 가지 인수를 사용합니다.

- SDIO 관련 인터럽트를 모니터링할 /proc 파일입니다. $irqnum은 mmc0 장치의 IRQ 번호입니다.

- OLED 디스플레이가 업데이트되는 속도(초당 횟수).
- SDIO 인터럽트 비율의 100%로 간주되는 초당 인터럽트 수입니다. 현재 수치는 시행착오를 거쳐 발견되었습니다.

boot 동안 이 유틸리티가 실행되지 않도록 하려면:

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

6.6 기타 참고 사항

- kernel 3.12 이후 Linux GPIO driver에 변경 사항이 있지만 Xillinux-1.3와 동일한 GPIO driver가 Xillinux-2.0에서 사용되어 Xillinux-1.3의 동작 및 번호 지정을 유지합니다.

  새 GPIO driver를 사용하려면 device tree 항목을 compatible = “xlnx,ps7-gpio-1.00.a”로 변경하여 “xlnx,zynq-gpio-1.0”로 표시합니다.

- driver가 쿼드 비트 인터페이스를 지원하더라도 Quad SPI flash는 1비트 너비의 bus를 사용하여 Linux kernel에 액세스합니다. 쿼드 비트 인터페이스를 활성화하기 위해 관련 device tree 항목을 변경할 수 있습니다. 이것은 동일한 device tree를 kernel 3.12(즉, Xillinux-1.3)와 함께 사용할 수 있도록 하는 기본 설정이 아닙니다.

7

문제 해결

7.1 implementation 중 오류

Xilinx 도구 릴리스 간의 약간의 차이로 인해 bitfile 생성을 위해 implementation을 실행하지 못하는 경우가 있습니다.

문제가 상당히 빨리 해결되지 않으면 Xillybus 포럼에서 도움을 요청하십시오.

http://forum.xillybus.com

특히 도구에서 보고한 첫 번째 오류 주변에 실패한 프로세스의 출력 로그를 첨부하십시오. 또한 design에서 사용자 지정 변경이 이루어진 경우 이러한 변경 사항을 자세히 설명하십시오. 또한 사용된 ISE/Vivado 도구의 버전도 알려주십시오.

VHDL용 implementation의 Vivado에서 이 오류가 발생하는 경우:

```
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 단락에 언급된 대로 xillydemo.vhd를 편집하십시오.

동일한 문제로 인해 발생할 수 있는 또 다른 오류:

```
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 키보드 및 마우스 문제

거의 모든 USB 키보드와 마우스는 호환 동작에 대한 표준 사양을 충족하므로 인식되지 않는 장치에서 문제가 발생할 가능성은 거의 없습니다. 문제가 발생하면 가장 먼저 확인해야 할 사항은 다음과 같습니다.

- Zedboard만 해당: 올바른 USB 플러그를 사용하고 있습니까? 전원 스위치에서 더 멀리 떨어진 “USB OTG”로 표시된 것이어야 합니다.
- Zedboard만 해당: 장치에 5V 공급 장치가 있습니까? JP2 점퍼가 설치되어 있습니까? Zedboard의 전원이 켜진 상태에서 광학 USB 마우스를 연결하고 LED가 켜져 있는지 확인합니다.
- USB hub을 사용하는 경우 키보드나 마우스만 보드에 직접 연결해 보십시오.

일반 시스템 로그 파일 /var/log/syslog에 유용한 정보가 있을 수 있습니다. “less /var/log/syslog”로 콘텐츠를 보는 것은 때때로 도움이 될 수 있습니다. 더욱이 “tail -f /var/log/syslog”을 입력하면 새 메시지가 도착할 때 console에 새 메시지가 덤프됩니다. 이는 특히 USB bus의 이벤트가 감지된 내용과 이벤트 처리 방법에 대한 자세한 설명을 포함하여 이 로그에 항상 기록되기 때문에 유용합니다.

shell prompt는 USB UART를 통해서도 액세스할 수 있으므로 키보드 연결에 실패할 경우 직렬 터미널로 로그를 볼 수 있습니다.

7.3 file system mount 관련 문제

경험에 따르면 적절한 (Micro)SD 카드를 사용하고 보드의 전원을 끄기 전에 시스템을 제대로 종료하면 (Micro)SD 카드의 데이터에 전혀 문제가 없습니다.

root file system의 unmounting 없이 보드의 전원을 끄면 ext4 file system이 다음 mount에서 journal로 자체 수리하기 때문에 file system 자체에서 영구적인 불일치를 일으키지 않을 것입니다. 그러나 전원이 꺼졌을 때 쓰기 위해 열린 파일이 잘못된 내용으로 남거나 완전히 삭제될 수 있기 때문에 운영 체제의 기능에 누적 손상이 있습니다. 이것은 갑자기 전원이 꺼진 컴퓨터의 경우에도 마찬가지입니다.

root file system이 mount를 수행하지 못하거나(boot 동안 kernel panic이 발생) mount를 read-only로만 수행하는 경우 가장 가능성이 높은 원인은 낮은 품질의 (Micro)SD 카드입

니다. 이러한 저장소가 잠시 동안 제대로 작동한 후 임의의 오류 메시지가 나타나기 시작하는 것은 매우 일반적입니다. /var/log/syslog에 다음과 같은 메시지가 포함되어 있으면 (Micro)SD 카드가 원인일 가능성이 큽니다.

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

이러한 문제를 방지하려면 Sandisk 장치를 고집하십시오.

7.4 “startx” 실패(Graphical desktop이 시작되지 않음)

직접적인 관련은 없지만 (Micro)SD 카드가 Sandisk에서 제작되지 않은 경우 이 문제가 자주 보고됩니다. 그래픽 소프트웨어는 시작할 때 카드에서 많은 양의 데이터를 읽으므로 읽기 오류를 생성하는 (Micro)SD 카드의 주목할만한 희생자가 될 수 있습니다.

확실한 해결책은 Sandisk (Micro)SD 카드를 사용하는 것입니다.

7.5 X desktop에서 화면 보호기 후 검은 화면

/root 디렉토리가 지워졌거나 새 사용자가 데스크탑을 사용 중이거나 절전 설정이 변경된 경우 데스크탑이 화면 보호기 모드에서 복구되지 않고 재개를 시도할 때 검은색 화면이 남을 수 있습니다.

수정: XFCE desktop에서 Power Manager로 이동하여 다음 설정을 지정합니다.

- Display > “Put to sleep after”를 “Never”로 설정
- Display > “Put to sleep after”를 “Never”로 설정
- Display > “Switch off after”를 “Never”로 설정합니다.
- Security > “Automatically lock the session”을 “Never”로 설정합니다.

이 설정은 이미 root 사용자에 대해 설정되어 있으므로 새로운 Xillinux 배포판에서는 이 문제가 발생하지 않아야 합니다.