PYNQ v3.1 ZYBO-Z7-20 보드 포팅 종합 가이드

PYNQ v3.1 포팅**:** 아키텍처 개요

PYNQ(Python Productivity for Zynq)를 새로운 하드웨어 보드에 포팅하는 작업은 단순히 일련의 명령어를 실행하는 것 이상의 과정입니다. 성공적인 포팅과 효율적인 문제 해결을 위해서는 PYNQ SD 카드 이미지의 근본적인 아키텍처를 이해하는 것이 필수적입니다. PYNQ 이미지는 본질적으로 두 개의 독립적인 파티션으로 구성되며, 각 파티션은 서로 다른 도구 체인에 의해 생성되고 고유한 역할을 수행합니다.

PYNQ 이미지의 이중 구조

PYNQ SD 카드 이미지는 하드웨어 종속적인 부팅 파티션과 아키텍처에 따라 거의 보드에 구애받지 않는 루트 파일 시스템 파티션으로 명확하게 구분됩니다. 이 구조적 분리는 포팅 과정에서 발생하는 문제의 원인을 체계적으로 진단하는 데 핵심적인 단서를 제공합니다.

부트 파티션 **(FAT32)**

이 파티션은 전적으로 Petalinux 도구 체인에 의해 생성됩니다.[1] 시스템을 기본적인 명령 프롬프트 상태까지 부팅시키는 데 필요한 모든 하드웨어 관련 구성 요소를 포함합니다. 주요 구성 요소는 다음과 같습니다.

- **FSBL (First Stage Bootloader):** Zynq SoC의 프로세서를 초기화하고 DDR 메모리와 같은 핵심 하드웨어를 설정합니다.
- **U-Boot (Second Stage Bootloader):** 전체 리눅스 커널을 메모리에 로드하고 실행을 시작하는 역할을 합니다.
- 리눅스 커널 **(image.ub):** 운영 체제의 핵심입니다.

- 디바이스 트리 블롭 **(system.dtb):** 부팅 시 커널에 하드웨어 구성을 설명하는 데이터 구조입니다.

이 파티션의 내용은 Vivado에서 생성된 하드웨어 정의 파일(.xsa)과 밀접하게 연결되어 있습니다.[3] 따라서 보드 부팅 과정에서 시리얼 콘솔에 아무런 출력이 없거나 U-Boot 프롬프트 이전에 멈추는 현상이 발생한다면, 문제의 원인은 거의 확실하게 부트 파티션 내의 구성 요소, 즉 Vivado 하드웨어 설계 오류나 Petalinux 설정 오류에 있습니다.

루트 파일 시스템 **(ext4)**

이 파티션은 Ubuntu 기반의 사용자 공간(user space)을 담고 있습니다.[1] 여기에는 PYNQ Python 라이브러리, Jupyter Notebook 서버, 각종 패키지 및 사용자가 상호작용하는 모든 애플리케이션이 포함됩니다. 이 파일 시스템은 동일한 프로세서 아키텍처(예: Zynq-7000 시리즈의 모든 보드는 ARMv7 아키텍처 사용) 내에서는 대부분 보드에 구애받지 않습니다. 보드가 성공적으로 부팅되어 리눅스 로그인 프롬프트가 나타났지만 이더넷과 같은 주변 장치가 작동하지 않거나 PYNQ 라이브러리에서 오류가 발생한다면, 이는 루트 파일 시스템의 설정 문제이거나, 더 흔하게는 커널 드라이버가 하드웨어에 올바르게 바인딩되지 못하도록 하는 디바이스 트리 구성 오류 때문일 가능성이 높습니다.

sdbuild 시스템의 역할

PYNQ sdbuild 시스템은 이러한 복잡한 과정을 조율하는 상위 수준의 프레임워크입니다.[5] 이는 단순한 스크립트가 아니라, Makefile 기반의 정교한 빌드 시스템으로, Petalinux를 호출하여 부트 파티션을 생성하고, chroot 환경을 구성하여 Ubuntu 루트 파일 시스템을 빌드하며, 이 모든 것을 최종적인 .img 파일로 패키징하는 과정을 자동화합니다.[7] 이 sdbuild 흐름에 ZYBO-Z7-20과 같은 특정 보드의 구성을 주입하는 핵심 메커니즘이 바로 "보드 저장소(board repository)"입니다.[5] 이 보고서의 후반부에서는 이 보드 저장소를 구성하는 방법을 단계별로 상세히 다룰 것입니다.

빌드 환경 구축

PYNQ 이미지 빌드 과정은 호스트 시스템의 환경에 매우 민감하며, 많은 빌드 실패가 부정확한 환경 설정에서 비롯됩니다. 따라서 공식적으로 검증된 안정적인 환경을 구축하는 것이 포팅

성공의 첫걸음입니다. PYNQ v3.1부터는 Docker를 이용한 빌드 방식이 표준으로 자리 잡았으며, 이는 이전 버전에서 빈번했던 환경 관련 문제를 획기적으로 줄여줍니다.[6]

호스트 시스템 및 **Xilinx** 도구 체인 요구사항

PYNQ 프레임워크는 특정 버전의 Xilinx 도구 체인과 강력한 의존성을 가집니다. 버전 불일치는 예측할 수 없는 빌드 오류의 주된 원인이므로, 아래 명시된 버전을 정확히 준수하는 것이 매우 중요합니다.

표 **1: PYNQ v3.1**을 위한 **Xilinx** 도구 체인 및 **OS** 호환성

구성 요소	요구 버전	비고
호스트 OS	Ubuntu 22.04 LTS (Jammy Jellyfish)	PYNQ v3.1 빌드 시스템이 공식적으로 지원하고 테스트된 OS입니다.[6]
PYNQ 버전	v3.1	이 보고서의 대상 버전입니다.
Vivado/Vitis	2024.1	PYNQ v3.1의 하드웨어 오버레이 및 플랫폼은 이 버전을 사용하여 빌드되었습니다.[6]
Petalinux	2024.1	PYNQ v3.1의 부트로더 및 커널 빌드는 이 버전을 사용합니다.[6]

이전 버전의 PYNQ 포팅 가이드에서 언급된 Vagrant 기반 가상 머신 환경은 PYNQ v3.1에서 더 이상 지원되지 않으며 Docker 기반 흐름으로 대체되었습니다.[8] 이는 복잡한 임베디드 빌드 시스템의 재현성을 보장하기 위해 컨테이너화 기술을 채택하는 업계 동향을 반영합니다. 수많은 포럼 게시물에서 확인되듯이, 과거에는 호스트 시스템의 미묘한 패키지 버전 차이나 설정 오류로 인해 빌드에 실패하는 경우가 많았습니다.[9] Docker는 이러한 모든 의존성을 격리된 컨테이너 안에 캡슐화함으로써, 누가 어떤 환경에서 실행하든 동일한 빌드 결과를 보장합니다.

권장 환경: **Docker** 기반 빌드 흐름

Docker를 사용하는 것은 선택 사항이 아닌, 안정적인 빌드를 위한 강력한 권장 사항입니다. 이 접근 방식은 시스템 의존성 및 도구 체인 구성을 자동으로 관리하여 환경 설정의 복잡성을 크게 줄여줍니다.

PYNQ 저장소 설정

빌드 환경이 준비되면, PYNQ 소스 코드를 올바르게 설정해야 합니다.

1. **PYNQ GitHub** 저장소 복제: 터미널을 열고 다음 명령어를 실행하여 공식 PYNQ 저장소를 복제합니다. --recursive 플래그는 sdbuild에 필요한 하위 모듈들을 함께 가져오기 위해 반드시 필요합니다.
   Bash
   git clone --recursive https://github.com/Xilinx/PYNQ.git

   12
2. 버전 태그 확인: 복제가 완료되면 PYNQ 디렉토리로 이동하여 v3.1에 해당하는 태그로 체크아웃합니다.
3. 호스트 설정 스크립트 실행: PYNQ/sdbuild 디렉토리로 이동하여 초기 호스트 설정 스크립트를 실행합니다. 이 스크립트는 qemu-user-static과 같이 빌드에 필요한 호스트 시스템 패키지를 설치합니다.
   Bash
   cd PYNQ/sdbuild

./scripts/setup_host.sh
```

7

이 단계까지 완료되면 PYNQ v3.1 이미지를 빌드하기 위한 모든 사전 준비가 끝납니다.

# 필수 포팅 아티팩트 목록
```

PYNQ 포팅은 단순히 파일 목록을 수집하는 것이 아니라, sdbuild 시스템이 인식할 수 있는 정형화된 **보드 저장소(board repository)**를 구축하는 과정입니다. 사용자가 이미 확보한 파일과 앞으로 생성해야 할 파일을 포함하여, ZYBO-Z7-20 포팅에 필요한 모든 아티팩트는 이 보드 저장소 구조 내에 위치하게 됩니다.

사용자 제공 파일 (기확보)

- **Vivado** 보드 정의 파일 **(board.xml, part0_pins.xml** 등**):** Vivado가 ZYBO-Z7-20 보드를 인식하고 프로젝트 생성 시 PS(Processing System) 설정, MIO 핀配置 등을 자동으로 구성하는 데 사용됩니다.[14]
- **Xilinx** 설계 제약 파일 **(.xdc):** 보드의 물리적 핀 위치, I/O 표준, 클럭 제약 등 하드웨어 레벨의 제약 조건을 정의합니다.[13]

하드웨어 생성 아티팩트 (생성 예정)

- **Xilinx** 셸 아카이브 **(.xsa):** 포팅 과정에서 가장 핵심적인 하드웨어 핸드오프 파일입니다. 이 파일은 Vivado 하드웨어 설계 전체를 담고 있는 컨테이너로, PS 설정, PL(Programmable Logic) 블록 다이어그램, 그리고 비트스트림까지 포함합니다. 이 파일은 Petalinux가 보드에 맞는 부트로더와 커널을 생성하는 데 사용하는 주된 입력이 됩니다.[3]

사용자 생성 구성 파일 (생성 예정)

- 보드 사양 파일 **(.spec):** sdbuild Makefile을 위한 빌드 레시피 파일입니다. 이 파일은 대상 보드의 아키텍처(ARM 또는 ARM64), 하드웨어 정의 파일(.xsa 또는 BSP)의 위치, 그리고 이미지에 포함될 보드별 추가 소프트웨어 패키지 목록을 정의합니다.[5]
- 디바이스 트리 소스 인클루드 **(system-user.dtsi):** 리눅스 커널에 ZYBO-Z7-20 보드의 고유한 오프칩(off-chip) 하드웨어(예: 이더넷 PHY 칩, 온보드 LED, 버튼 등)를 설명하는 데 사용됩니다. Petalinux는 .xsa 파일로부터 Zynq SoC 내부의 하드웨어만 인식할 수 있으므로, 이 파일을 통해 보드 레벨의 하드웨어 정보를 추가로 제공해야 합니다.[11]

아래 표는 포팅에 필요한 모든 파일과 아티팩트를 요약한 체크리스트입니다. 각 항목의 역할과 출처, 그리고 PYNQ 저장소 내 최종 위치를 명확히 하여 전체 과정을 체계적으로 관리할 수 있도록 돕습니다.

표 **2: ZYBO-Z7-20** 포팅을 위한 필수 파일 및 아티팩트 체크리스트

아티팩트 이름	목적	출처	PYNQ 저장소 내 목표 위치
Vivado 보드 파일	Vivado가 보드를 인식하고 프로젝트를 사전 구성	Digilent 제공	<Vivado_설치경로>/data/boards/board_files/
.xdc 파일	물리적 핀 할당 및 I/O 표준 정의	Digilent 제공	Vivado 프로젝트 소스
Zybo-Z7-20.xsa	전체 하드웨어 설계를 Petalinux에 전달	Vivado에서 생성	sdbuild/boards/Zybo-Z7-20/petalinux_bsp/hardware_project/
Zybo-Z7-20.spec	sdbuild를 위한 빌드 레시피 정의	사용자 생성	sdbuild/boards/Zybo-Z7-20/
system-user.dtsi	커널에 보드 레벨 하드웨어(이더넷 PHY 등) 설명	사용자 생성	sdbuild/boards/Zybo-Z7-20/petalinux_bsp/meta-user/recipes-bsp/device-tree/files/

1단계: **Vivado**를 이용한 하드웨어 플랫폼 생성

포팅 과정의 첫 번째 실질적인 단계는 Vivado를 사용하여 ZYBO-Z7-20 보드를 위한 기본 하드웨어 플랫폼을 설계하고, 이를 Petalinux가 사용할 수 있는 형식인 .xsa 파일로 내보내는 것입니다. 이 .xsa 파일은 단순한 비트스트림이 아니라, PS 설정, 메모리 맵, 클럭 정보 등 Petalinux가 부트로더와 디바이스 트리를 자동 생성하는 데 필요한 모든 정보를 담고 있는 포괄적인 하드웨어 데이터베이스입니다. 따라서 이 단계에서의 정확성은 전체 포팅 과정의 성공을 좌우하는 기반이 됩니다.

프로젝트 생성

1. **Vivado** 실행**:** PYNQ v3.1과 호환되는 **Vivado 2024.1** 버전을 실행합니다.
2. 새 프로젝트 생성**:** "Create Project"를 선택하여 새 RTL 프로젝트를 시작합니다. 프로젝트 이름과 경로를 지정한 후, "Boards" 탭에서 ZYBO-Z7-20을 선택합니다.[15] 이 목록에 보드가 나타나려면 사전에 Digilent에서 제공하는 보드 정의 파일을 Vivado 설치 경로에 복사해 두어야 합니다.

블록 디자인 및 **PS** 구성

1. 블록 디자인 생성**:** IP Integrator 메뉴에서 "Create Block Design"을 선택하여 새로운 블록 다이어그램을 생성합니다.
2. **Zynq PS IP** 추가**:** 다이어그램에 "ZYNQ7 Processing System" IP 코어를 추가합니다.[15]
3. 블록 자동화 실행**:** IP가 추가되면 상단에 나타나는 녹색 배너에서 "Run Block Automation"을 클릭합니다. 이 기능은 보드 정의 파일에 미리 설정된 값들을 Zynq PS에 자동으로 적용하여 DDR, MIO 주변 장치, 클럭 등을 구성합니다.[22]
4. 핵심 검증 단계**:** 자동 구성이 완료된 후, Zynq PS IP를 더블 클릭하여 설정 창을 엽니다. "DDR Configuration"과 "Clock Configuration" 탭의 값들이 ZYBO-Z7-20의 공식 사양(데이터시트)과 일치하는지 반드시 수동으로 확인해야 합니다. DDR 타이밍이나 클럭 주파수 설정의 미세한 불일치는 부팅이 조용히 실패하는(silent boot failure) 주된 원인이 됩니다.[13]

최소한의 **PL** 디자인 **(**선택 사항이지만 권장**)**

PYNQ의 기본 포팅을 위해서는 PL에 추가적인 로직이 반드시 필요한 것은 아닙니다. 하지만 초기 테스트 및 검증을 위해 보드 위의 LED와 버튼을 제어하는 AXI GPIO 컨트롤러를 추가하는 것이 좋은 방법입니다.[15] 이는 나중에 PYNQ 환경에서 간단한 Python 코드로 하드웨어가 올바르게 프로그래밍되었는지 확인하는 데 유용합니다.

합성**,** 구현 및 하드웨어 내보내기

1. **HDL** 래퍼 생성**:** 블록 디자인을 완성한 후, "Sources" 창에서 블록 디자인 파일(.bd)을 마우스 오른쪽 버튼으로 클릭하고 "Create HDL Wrapper"를 선택합니다.
2. 비트스트림 생성**:** "Generate Bitstream"을 클릭하여 합성(Synthesis)과 구현(Implementation)을 거쳐 최종 비트스트림 파일을 생성합니다.
3. 하드웨어 내보내기 **(**가장 중요한 단계**):** 메뉴에서 File -> Export -> Export Hardware를 선택합니다. 대화 상자에서 플랫폼 유형을 "Fixed"로 선택하고, 출력 형식이 "XSA"인지 확인한 후, **"Include bitstream"** 옵션이 반드시 체크되어 있는지 확인합니다. 이 과정을 통해 Zybo-Z7-20.xsa 파일이 생성됩니다.[3]

이 .xsa 파일은 Vivado 단계의 최종 산출물이며, 다음 단계인 Petalinux 기반의 소프트웨어 빌드 과정의 핵심 입력 자료로 사용됩니다. Vivado PS 설정의 오류는 Petalinux에 의해 자동으로 디바이스 트리와 부트로더에 전파되므로, 이 단계에서 하드웨어 구성을 꼼꼼히 검증하는 것은 후반부의 디버깅 시간을 크게 절약하는 지름길입니다.

2단계**: ZYBO-Z7-20** 보드 저장소 구축

sdbuild Makefile은 고도로 관례화된(convention-driven) 방식으로 동작합니다. 즉, 특정 이름과 구조를 가진 디렉토리와 파일을 예상하며, 이 규칙을 따르지 않으면 빌드 프로세스가 실패합니다. 이 단계에서는 sdbuild 시스템이 ZYBO-Z7-20 보드를 인식하고 빌드할 수 있도록 필요한 디렉토리 구조를 만들고, 핵심 구성 파일들을 배치하는 방법을 설명합니다.

디렉토리 구조 생성

1. 이전에 복제한 PYNQ/ 저장소의 최상위 디렉토리에서 시작합니다.
2. sdbuild/boards/ 경로 안에 Zybo-Z7-20이라는 새 디렉토리를 생성합니다. 이 디렉토리 이름은 매우 중요하며, 이후 make 명령어에서 BOARDS 변수로 사용될 이름과 정확히 일치해야 합니다.[5]
3. 새로 만든 Zybo-Z7-20/ 디렉토리 안에 petalinux_bsp/hardware_project/ 경로를 순서대로 생성합니다.[17]

완성된 디렉토리 구조는 다음과 같아야 합니다.

PYNQ/

```
└── sdbuild/
    └── boards/
        └── Zybo-Z7-20/
            └── petalinux_bsp/
                └── hardware_project/
```

하드웨어 정의 파일 배치

이전 1단계에서 Vivado를 통해 생성한 Zybo-Z7-20.xsa 파일을 위에서 만든 PYNQ/sdbuild/boards/Zybo-Z7-20/petalinux_bsp/hardware_project/ 디렉토리 안으로 복사합니다.[13]

sdbuild 시스템은 이 위치에서 .xsa 파일을 찾아 Petalinux BSP(Board Support Package)를 자동으로 생성하는 데 사용합니다.

.spec 파일 작성

sdbuild Makefile을 위한 빌드 레시피인 .spec 파일을 작성해야 합니다.

1. PYNQ/sdbuild/boards/Zybo-Z7-20/ 디렉토리 안에 Zybo-Z7-20.spec이라는 이름의 새 텍스트 파일을 생성합니다. 파일 이름은 보드 디렉토리 이름과 동일해야 합니다.
2. 파일을 열고 아래 내용을 입력합니다. 각 변수는 주석을 통해 설명되어 있습니다.[5]

RPM spec files

```
# Zynq-7000 디바이스의 아키텍처는 'arm'입니다.
ARCH_Zybo-Z7-20 := arm

# BSP 변수는 비워둡니다. 빌드 시스템이 petalinux_bsp/hardware_project/ 안의.xsa 파일로부터 BSP를 생성할 것입니다.
BSP_Zybo-Z7-20 :=

# 선택 사항: 기본 비트스트림을 지정합니다. 비워두면.xsa에 포함된 비트스트림이 사용됩니다.
BITSTREAM_Zybo-Z7-20 :=
```

PYNQ 코어 패키지와 네트워킹을 위한 이더넷 패키지를 최종 이미지에 설치합니다.
STAGE4_PACKAGES_Zybo-Z7-20 := pynq ethernet

이 과정에서 명명 규칙의 일관성을 유지하는 것이 매우 중요합니다. make BOARDS=Zybo-Z7-20 명령을 실행하면, Makefile은 sdbuild/boards/ 디렉토리에서 Zybo-Z7-20 폴더를 찾고, 그 안에서 Zybo-Z7-20.spec 파일을 읽어 ARCH_Zybo-Z7-20과 같은 변수들을 찾습니다. 디렉토리 이름, .spec 파일 이름, 그리고 .spec 파일 내부의 변수 접두사가 모두 일치하지 않으면 변수가 정의되지 않았다는 오류와 함께 빌드가 실패하게 됩니다. 이는 초보자들이 흔히 겪는 문제이므로, 제공된 템플릿을 정확히 따르는 것이 중요합니다.

3단계**:** 리눅스 커널을 위한 디바이스 트리 구성

이 단계는 포팅 과정에서 기술적으로 가장 섬세한 부분으로, ZYBO-Z7-20 보드의 하드웨어 구성을 리눅스 커널에 정확하게 알려주는 작업입니다. 디바이스 트리(Device Tree)는 하드웨어 설명을 커널 소스 코드로부터 분리하는 데이터 구조로, 이를 통해 동일한 커널 이미지를 다양한 하드웨어 구성에서 사용할 수 있게 합니다.[19]

디바이스 트리와 **PYNQ/Petalinux** 빌드 흐름

PYNQ의 빌드 시스템에서 디바이스 트리는 다음과 같은 과정을 통해 생성됩니다.

1. Vivado에서 생성된 .xsa 파일은 Zynq SoC 내부의 IP(예: 이더넷 MAC, UART, I2C 컨트롤러)에 대한 정보를 담고 있습니다.
2. Petalinux는 이 .xsa 파일을 파싱하여 pl.dtsi, pcw.dtsi 등 기본 디바이스 트리 소스 파일들을 자동으로 생성합니다.
3. 하지만 .xsa 파일에는 보드 위에 장착된 외부 칩(예: 이더넷 PHY, 오디오 코덱)에 대한 정보가 없습니다.
4. 사용자는 system-user.dtsi라는 파일을 통해 이 누락된 하드웨어 정보를 추가하거나, 자동 생성된 설정을 변경(override)해야 합니다.[19]

system-user.dtsi 파일 생성 및 배치

1. 이전 단계에서 만든 보드 저장소 내에 다음 경로를 생성합니다.

PYNQ/sdbuild/boards/Zybo-Z7-20/petalinux_bsp/meta-user/recipes-bsp/device-tree/files/

2. 이 files/ 디렉토리 안에 system-user.dtsi라는 이름의 새 파일을 생성합니다.

ZYBO-Z7-20을 위한 system-user.dtsi 작성

ZYBO-Z7-20에서 네트워크 기능을 활성화하기 위해 가장 중요한 것은 이더넷 PHY 칩(Realtek RTL8211E)에 대한 정보를 커널에 제공하는 것입니다. 아래는 이를 위한 system-user.dtsi의 예시 코드입니다. 이 코드를 위에서 생성한 파일에 복사하여 붙여넣습니다.

DTS

```
/include/ "system-conf.dtsi"
/ {
    /* 만약 Vivado에서 LED나 버튼을 위한 AXI GPIO를 추가했다면,
       여기에 해당 노드를 추가할 수 있습니다. 예:
    aliases {
        gpio0 = &axi_gpio_0;
    };
    */
};

/* Zynq의 내장 기가비트 이더넷 MAC(gem0)에 대한 설정을 수정합니다. */
&gem0 {
    /* 이 MAC이 사용할 PHY의 핸들을 지정합니다. 아래의 phy0을 가리킵니다. */
    phy-handle = <&phy0>;
    /* RGMII-ID 모드를 사용하도록 설정합니다. PHY가 내부 지연을 제공합니다. */
    phy-mode = "rgmii-id";

    /* gem0에 연결된 MDIO 버스를 정의합니다. */
    ps7_ethernet_0_mdio: mdio {
        #address-cells = ;
        #size-cells = ;

        /* MDIO 버스에 연결된 PHY 칩을 정의합니다. */
        /* ZYBO-Z7의 Realtek PHY는 주소 1번을 사용합니다. */
        phy0: phy@1 {
            compatible = "realtek,rtl8211e";
```

```
        device_type = "ethernet-phy";
        reg = ;
      };
    };
};
```

11

코드 설명

- &gem0: 이 구문은 새로운 노드를 정의하는 것이 아니라, Petalinux가 자동 생성한 gem0 (Zynq의 기가비트 이더넷 MAC) 노드를 참조하여 그 내용을 수정하겠다는 의미입니다. 이는 디바이스 트리 오버레이의 핵심적인 메커니즘입니다.
- phy-handle = <&phy0>;: gem0 MAC이 통신할 PHY 디바이스가 phy0임을 명시합니다.
- phy@1: MDIO 버스 상에서 주소(address)가 1인 디바이스를 의미합니다. ZYBO-Z7-20 보드 회로도에 따르면 이더넷 PHY 칩은 주소 1에 할당되어 있습니다. 이 주소 값이 틀리면 커널이 PHY 칩을 찾지 못해 네트워크가 동작하지 않습니다.[11]
- compatible = "realtek,rtl8211e";: 리눅스 커널이 이 디바이스에 어떤 드라이버를 사용해야 하는지를 알려주는 "호환성 문자열"입니다. 커널은 이 문자열과 일치하는 드라이버를 찾아 로드합니다.

이처럼 system-user.dtsi 파일은 자동 생성된 하드웨어 정보의 불완전한 부분을 보충하는 "패치" 역할을 합니다. .xsa에 없는 외부 하드웨어 정보를 정확히 기술함으로써, 커널 드라이버가 부팅 시점에 해당 하드웨어를 성공적으로 인식하고 초기화할 수 있도록 만들어 줍니다.

4단계: PYNQ SD 카드 이미지 빌드

모든 하드웨어 정의와 구성 파일이 준비되었으므로, 이제 sdbuild Makefile을 실행하여 최종 PYNQ SD 카드 이미지를 빌드할 차례입니다. 이 과정은 수 시간이 소요될 수 있으며, 시스템 사양에 따라 달라집니다.

환경 스크립트 소싱(Sourcing)

make 명령어를 실행하기 전에, 현재 터미널 세션에 Xilinx 도구 체인의 경로가 설정되어 있어야 합니다. 이는 빌드 시스템이 Vivado, Vitis, Petalinux 실행 파일을 찾을 수 있도록 하기 위함입니다.

Bash

```
source <Vitis_설치경로>/Vitis/2024.1/settings64.sh
source <Petalinux_설치경로>/petalinux-2024.1-final/settings.sh
```

6

빌드 실행

1. **sdbuild** 디렉토리로 이동**:** 터미널에서 PYNQ/sdbuild 디렉토리로 이동합니다.
2. 빌드 명령어 실행**:** 다음 명령어를 입력하여 ZYBO-Z7-20 보드에 대한 빌드를 시작합니다.
   Bash
   ```
   make BOARDS=Zybo-Z7-20
   ```
   12

이 명령어를 실행하면 sdbuild 시스템은 다음과 같은 일련의 작업을 자동으로 수행합니다.

- BOARDS 변수에 지정된 Zybo-Z7-20 보드 저장소를 찾습니다.
- Zybo-Z7-20.spec 파일을 읽어 아키텍처와 패키지 정보를 확인합니다.
- petalinux_bsp/hardware_project/ 디렉토리의 .xsa 파일을 사용하여 Petalinux 프로젝트를 생성하고 BSP를 빌드합니다. 이 과정에서 FSBL, U-Boot, 커널, 디바이스 트리가 컴파일됩니다.
- 미리 빌드된 보드 불특정(board-agnostic) ARM 루트 파일 시스템(pynq_rootfs.arm.tar.gz)과 PYNQ 소스 배포판(pynq_sdist.tar.gz)을 다운로드하여 사용합니다. 이는 전체 빌드 시간을 몇 시간 단축시키는 매우 효율적인 최적화입니다.[1]
- chroot와 qemu를 사용하여 ARM 환경을 에뮬레이션하고, 루트 파일 시스템 내에 .spec 파일에 명시된 pynq와 ethernet 같은 패키지들을 설치합니다.[7]
- Petalinux에서 생성된 부트 파티션과 패키지가 설치된 루트 파일 시스템을 결합하여 최종 .img 파일을 생성합니다.

빌드 실패 시 복구

sdbuild Makefile은 빌드 속도를 높이기 위해 중간 결과물들을 캐싱합니다.[27] 만약 빌드가 중간에 실패하면, 이 캐시된 파일들이나 마운트된 루프백 디바이스가 비정상적인 상태로 남을 수 있습니다. 이 상태에서 단순히

make를 다시 실행하면 동일한 오류가 발생하거나 다른 예측 불가능한 문제에 부딪힐 수 있습니다.

이러한 상황에서는 다음 복구 명령어를 사용하는 것이 필수적입니다.

- 마운트 해제: 빌드 중 생성된 루프백 마운트를 정리합니다.
  Bash
  ```
  make unmount
  ```
- 빌드 초기화: 모든 중간 빌드 아티팩트와 캐시를 삭제하고 완전히 처음부터 빌드를 시작하고 싶을 때 사용합니다.
  Bash
  ```
  make clean
  ```

  6

빌드 실패 시에는 먼저 make unmount나 make clean을 실행하여 빌드 환경을 깨끗하게 정리한 후, 오류 메시지를 분석하고 원인을 수정한 뒤 다시 빌드를 시도하는 것이 체계적인 접근 방식입니다.

최종 결과물 확인

빌드가 성공적으로 완료되면, 최종적으로 부팅 가능한 SD 카드 이미지는 PYNQ/sdbuild/output/ 디렉토리 안에 Zybo-Z7-20.img라는 이름으로 생성됩니다.[12]

배포, 검증 및 문제 해결

최종 .img 파일이 생성되면, 이를 SD 카드에 쓰고 보드에서 부팅하여 포팅 결과를 검증해야 합니다. 이 과정에서 발생할 수 있는 일반적인 문제들과 그 해결 방법을 체계적으로 정리하여

제공합니다.

SD 카드 이미지 작성 및 보드 설정

1. **SD** 카드 이미지 작성**:** Balena Etcher, Win32DiskImager 또는 Linux의 dd 명령어와 같은 도구를 사용하여 Zybo-Z7-20.img 파일을 최소 16GB 용량의 microSD 카드에 기록합니다.[28]
2. 보드 하드웨어 설정**:**
   - 부트 모드 점퍼 **(ZYBO-Z7**의 **JP5):** 점퍼를 "SD" 위치로 설정하여 microSD 카드로부터 부팅하도록 지정해야 합니다. 이는 자주 간과되는 물리적 설정 단계입니다.[30]
   - 전원 소스 점퍼 **(JP6):** 사용하는 전원(USB 또는 외부 어댑터)에 맞게 점퍼를 설정합니다. 복잡한 PL 설계를 사용하거나 여러 주변 장치를 연결할 경우 USB 전원만으로는 전류가 부족할 수 있으므로 외부 전원 사용이 권장됩니다.[26]
   - 케이블 연결**:** 전원, 이더넷 케이블, 그리고 디버깅을 위한 UART용 Micro-USB 케이블을 보드에 연결합니다.

초기 부팅 및 검증

1. 시리얼 터미널 연결**:** PC에서 PuTTY, minicom, Tera Term과 같은 터미널 에뮬레이터를 실행하고, ZYBO-Z7-20의 UART에 해당하는 COM 포트에 연결합니다. 통신 설정은 **115200 baud, 8** 데이터 비트**,** 패리티 없음**, 1** 정지 비트 **(8N1)** 입니다.[13]
2. 부팅 및 로그 확인**:** 보드의 전원을 켜고 시리얼 터미널에 출력되는 부팅 로그를 관찰합니다. U-Boot 메시지로 시작하여 리눅스 커널 부팅 메시지가 이어지고, 최종적으로 pynq login: 프롬프트가 나타나면 성공적으로 부팅된 것입니다.
3. 로그인 및 네트워크 확인**:** 기본 사용자 이름 xilinx와 비밀번호 xilinx로 로그인합니다. 로그인 후 ifconfig 또는 ip addr 명령어로 eth0 인터페이스가 IP 주소를 할당받았는지 확인하고, ping 명령어로 외부 네트워크와의 연결을 테스트합니다.
4. **Jupyter Notebook** 접속**:** PC의 웹 브라우저 주소창에 http://<보드의_IP_주소>:9090을 입력하여 Jupyter Notebook 서버에 접속합니다. 기본 비밀번호는 xilinx입니다.

체계적인 문제 해결 가이드

포팅 과정에서 문제가 발생했을 때, 증상에 따라 원인을 좁혀 나갈 수 있습니다.

- 증상**:** 시리얼 콘솔에 아무런 출력이 없음**.**

    - 가능한 원인:
        - 부트 모드 점퍼(JP5) 설정 오류 (SD 모드가 아닌 경우).
        - 불충분한 전원 공급.
        - SD 카드가 올바르게 포맷되지 않았거나 이미지가 손상된 경우.
        - Vivado에서 PS 구성(특히 클럭 또는 DDR)이 잘못되어 FSBL 실행에 실패하는 경우.
- 증상: 부팅이 시작되지만 커널 로딩 중 멈추거나 커널 패닉(**Kernel Panic**) 발생.
    - 가능한 원인:
        - Vivado에서 DDR 메모리 타이밍이나 파라미터 설정이 ZYBO-Z7-20 보드와 일치하지 않는 경우.
        - 디바이스 트리(system-user.dtsi)에 치명적인 오류가 있어 필수 드라이버 로딩에 실패하는 경우.
- 증상: 리눅스 로그인까지는 성공하지만, 이더넷/네트워크가 작동하지 않음.
    - 가능한 원인:
        - 거의 대부분 system-user.dtsi 파일의 이더넷 PHY 설정 오류입니다. PHY의 MDIO 주소(reg = ;), 호환성 문자열(compatible), 또는 phy-mode가 잘못되었을 가능성이 높습니다.[11]
- 증상: **make** 빌드 명령어가 중간에 실패함.
    - 가능한 원인:
        - 호스트 PC에 설치된 Xilinx 도구 체인(Vivado, Petalinux) 버전이 PYNQ v3.1 요구사항(2024.1)과 다른 경우.
        - ./scripts/setup_host.sh 실행이 실패했거나 필요한 호스트 의존성 패키지가 누락된 경우.
        - (드물게) Vivado 디자인에 포함된 특정 IP에 대한 라이선스가 없는 경우.[11]
        - 이전 빌드 실패 후 make clean 또는 make unmount를 실행하지 않아 빌드 환경이 오염된 경우.

결론

본 보고서는 PYNQ v3.1을 Digilent ZYBO-Z7-20 보드에 포팅하는 전 과정을 체계적으로 안내했습니다. 성공적인 포팅은 단순히 명령어를 나열하는 것이 아니라, PYNQ 이미지의 아키텍처적 이해를 바탕으로 각 단계의 산출물이 다음 단계에 미치는 영향을 파악하는 데서 출발합니다.

핵심적인 성공 요인은 다음과 같이 요약할 수 있습니다.

1. 정확한 빌드 환경 구축: PYNQ v3.1은 Vivado/Petalinux 2024.1 버전 및 Ubuntu 22.04 호스트 OS와 엄격한 의존성을 가집니다. 공식적으로 권장되는 Docker 기반 빌드 환경을 사용하는 것이 환경 변수로 인한 실패를 최소화하는 가장 확실한 방법입니다.
2. 견고한 하드웨어 플랫폼 설계: 모든 소프트웨어 구성의 기초가 되는 것은 Vivado에서

생성된 .xsa 파일입니다. Zynq PS의 DDR 및 클럭 설정을 보드 사양과 정확히 일치시키는 과정은 부팅 실패를 방지하는 데 가장 중요합니다.

3. 체계적인 보드 저장소 구성: PYNQ의 sdbuild 시스템은 명명 규칙과 디렉토리 구조에 의존합니다. Zybo-Z7-20이라는 보드 이름을 디렉토리, .spec 파일, 그리고 그 안의 변수들에 일관되게 적용하는 것이 필수적입니다.
4. 정밀한 디바이스 트리 수정: 리눅스 커널이 보드 레벨의 하드웨어, 특히 이더넷 PHY와 같은 외부 칩을 인식하게 하는 system-user.dtsi 파일의 작성은 포팅의 성패를 가르는 핵심 기술입니다. PHY의 주소와 호환성 문자열을 정확히 명시하는 것이 네트워크 기능을 활성화하는 데 결정적입니다.

사용자는 제공된 board.xml 및 .xdc 파일을 시작으로, 본 가이드에 따라 단계별로 .xsa 파일, Zybo-Z7-20.spec 파일, 그리고 system-user.dtsi 파일을 생성하고 올바른 위치에 배치해야 합니다. 이 아티팩트들이 모두 준비되면, make BOARDS=Zybo-Z7-20 명령어를 통해 최종 부팅 이미지를 생성할 수 있습니다. 마지막으로, 부팅 과정에서 문제가 발생할 경우, 시리얼 콘솔 로그를 면밀히 분석하고 본 보고서의 문제 해결 가이드를 참조하여 체계적으로 원인을 진단하는 것이 권장됩니다. 이 과정을 통해 ZYBO-Z7-20 보드에서 PYNQ v3.1의 강력한 Python 기반 하드웨어 가속 기능을 완전히 활용할 수 있게 될 것입니다.

참고 자료

1. Quick Porting of PYNQ using Pre-built Images - Learn, 9월 26, 2025에 액세스, https://discuss.pynq.io/t/quick-porting-of-pynq-using-pre-built-images/1075
2. Porting PYQN to ZC702 - element14 Community, 9월 26, 2025에 액세스, https://community.element14.com/technologies/fpga-group/b/blog/posts/porting-pyqn-to-zc702
3. PYNQ-Z2 and Petalinux : r/FPGA - Reddit, 9월 26, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/1b6gr76/pynqz2_and_petalinux/
4. Porting PYNQ to ZC702 - Hackster.io, 9월 26, 2025에 액세스, https://www.hackster.io/dhq/porting-pynq-to-zc702-218809
5. sdbuild · v2.6.0 · paulrr2 / PYNQ - GitLab at Illinois, 9월 26, 2025에 액세스, https://gitlab.engr.illinois.edu/paulrr2/PYNQ/-/tree/v2.6.0/sdbuild
6. PYNQ SD Card image — Python productivity for Zynq (Pynq) - Read the Docs, 9월 26, 2025에 액세스, https://pynq.readthedocs.io/en/latest/pynq_sd_card.html
7. sdbuild · image_v2.2 · paulrr2 / PYNQ - GitLab at Illinois, 9월 26, 2025에 액세스, https://gitlab-03.engr.illinois.edu/paulrr2/PYNQ/-/tree/image_v2.2/sdbuild?ref_type=heads
8. Releases · Xilinx/PYNQ - GitHub, 9월 26, 2025에 액세스, https://github.com/Xilinx/PYNQ/releases
9. PYNQ 3.0.1 VirtualBox install and SD build - Learn, 9월 26, 2025에 액세스, https://discuss.pynq.io/t/pynq-3-0-1-virtualbox-install-and-sd-build/4859
10. Novel Issues: Quick Porting of PYNQ using Pre-built images - Support, 9월 26, 2025에 액세스, https://discuss.pynq.io/t/novel-issues-quick-porting-of-pynq-using-pre-built-images/4187
11. PYNQ 2.7 custom board building error - Support, 9월 26, 2025에 액세스,

https://discuss.pynq.io/t/pynq-2-7-custom-board-building-error/4649
12. Eclypse-Z7 PYNQ porting guide - Hackster.io, 9월 26, 2025에 액세스, https://www.hackster.io/bartosz-rycko/eclypse-z7-pynq-porting-guide-3dd24c
13. Pynq 2.6 custom board image build method that works - Learn, 9월 26, 2025에 액세스, https://discuss.pynq.io/t/pynq-2-6-custom-board-image-build-method-that-works/2894
14. xupsh/pynq-supported-board-file - GitHub, 9월 26, 2025에 액세스, https://github.com/xupsh/pynq-supported-board-file
15. Tutorial: Creating a hardware design for PYNQ - Learn, 9월 26, 2025에 액세스, https://discuss.pynq.io/t/tutorial-creating-a-hardware-design-for-pynq/145
16. PYNQ Build Image for a Custom Board : r/FPGA - Reddit, 9월 26, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/1lsep3h/pynq_build_image_for_a_custom_board/
17. Creating Pynq images for custom boards. - controlpaths.com, 9월 26, 2025에 액세스, Creating Pynq images for custom boards. | controlpaths.com
18. PYNQ sd card for custom board fails to boot - Support, 9월 26, 2025에 액세스, https://discuss.pynq.io/t/pynq-sd-card-for-custom-board-fails-to-boot/147
19. MicroZed Chronicles: Device Trees - Hackster.io, 9월 26, 2025에 액세스, https://www.hackster.io/news/microzed-chronicles-device-trees-92566170bb51
20. PetaLinux Building and System Customization — Vitis™ Tutorials 2022.1 documentation, 9월 26, 2025에 액세스, https://xilinx.github.io/Vitis-Tutorials/2022-1/build/html/docs/Vitis_Platform_Creation/Feature_Tutorials/02_petalinux_customization/README.html
21. Starter Lab With the Block Diagram, 9월 26, 2025에 액세스, https://jodalyst.com/6s193/pynq_lab1/
22. PYNQ Edition! Creating Custom Overlays - Hackster.io, 9월 26, 2025에 액세스, https://www.hackster.io/news/pynq-edition-creating-custom-overlays-8543f45eccb1
23. AxiGPIO — Python productivity for Zynq (Pynq) v1.0 - Read the Docs, 9월 26, 2025에 액세스, https://pynq.readthedocs.io/en/v2.4/pynq_libraries/axigpio.html
24. PYNQ SD Card — Python productivity for Zynq (Pynq) - Read the Docs, 9월 26, 2025에 액세스, https://pynq.readthedocs.io/en/v2.5.1/pynq_sd_card.html
25. Zybo Z7 Petalinux Demo - Digilent Reference, 9월 26, 2025에 액세스, https://digilent.com/reference/programmable-logic/zybo-z7/demos/petalinux
26. Digilent/Petalinux-Zybo-Z7-20 - GitHub, 9월 26, 2025에 액세스, https://github.com/Digilent/Petalinux-Zybo-Z7-20
27. Help understanding output files from sdbuild/make - #8 by PeterOgden - Support - PYNQ, 9월 26, 2025에 액세스, https://discuss.pynq.io/t/help-understanding-output-files-from-sdbuild-make/191/8
28. Define Custom Pynq-Z2 Board with SoC Blockset - MATLAB & Simulink - MathWorks, 9월 26, 2025에 액세스, https://www.mathworks.com/help/soc/ug/define-custom-pynq-z2-board-with-socb.html

29. Getting Started — Python productivity for Zynq (Pynq) v1.0 - Read the Docs, 9월 26, 2025에 액세스, https://pynq.readthedocs.io/en/v2.1/getting_started.html
30. PYNQ-Z1 Setup Guide — Python productivity for Zynq (Pynq) - Read the Docs, 9월 26, 2025에 액세스, https://pynq.readthedocs.io/en/v2.7.0/getting_started/pynq_z1_setup.html
31. PYNQ-Z1 Reference Manual - Digilent, 9월 26, 2025에 액세스, https://digilent.com/reference/programmable-logic/pynq-z1/reference-manual
32. PYNQ-Z1 Board Reference Manual Overview - Digilent, 9월 26, 2025에 액세스, https://digilent.com/reference/_media/reference/programmable-logic/pynq-z1/pynq-rm.pdf