ZYNQ7020 플랫폼을 위한 **PYNQ 3.1** 경량 이미지 빌드 심층 분석 보고서

서론**: ZYNQ7020** 플랫폼을 위한 **PYNQ** 경량화의 도전 과제

본 보고서는 PYNQ 버전 3.1 환경에서 pynqremote 빌드 옵션을 사용하여 ZYNQ7020 시스템온칩(SoC) 기반 보드(예: PYNQ-Z1, PYNQ-Z2)용으로 200MB 미만의 초경량 SD 카드 이미지를 생성한 성공 사례를 찾는 기술 질의에 대한 심층 분석 결과를 제공합니다. 이 질의는 단순한 정보 검색을 넘어, 리소스가 제한된 임베디드 환경에서 PYNQ 프레임워크의 적용 가능성을 타진하는 핵심적인 기술적 과제를 담고 있습니다. 표준 PYNQ 이미지가 수 기가바이트(GB)에 달하는 상황에서 [1], 이러한 수준의 경량화는 상용 제품화 또는 특정 연구 분야로의 적용을 위한 필수 전제 조건이 될 수 있습니다.

광범위한 자료 분석 결과, 사용자의 요구사항(PYNQ 3.1, pynqremote 빌드, ZYNQ7020, 200MB 이하)을 모두 만족하는 검증된 공식 성공 사례나 관련 링크는 현재까지 발견되지 않았습니다.

따라서 본 보고서는 단순한 검색 결과의 부재를 통보하는 것을 넘어, 해당 경로가 왜 기술적으로 어려운지에 대한 근본적인 원인을 분석하고, 사용자의 최종 목표를 달성하기 위한 가장 현실적이고 신뢰성 높은 대안 엔지니어링 경로를 제시하는 것을 목표로 합니다. 보고서는 PYNQ 이미지의 구조적 특성, pynqremote 빌드의 실체, 200MB 목표의 현실성 평가, 그리고 최종적으로 실행 가능한 대안 전략 제시 순으로 구성됩니다.

제**1**장**: PYNQ** 이미지의 구조와 크기 결정 요인 분석

1.1 PYNQ의 설계 철학**:** 생산성 우선주의

PYNQ 프레임워크는 Python 언어와 Jupyter 노트북이라는 대화형 환경을 통해

FPGA(Field-Programmable Gate Array) 개발의 생산성을 극대화하는 것을 최우선 목표로 설계되었습니다.[3] 이는 개발자가 복잡한 하드웨어 기술에 대한 깊은 이해 없이도 소프트웨어 개발과 유사한 방식으로 FPGA의 병렬 처리 능력을 활용할 수 있도록 지원하기 위함입니다. 이러한 설계 철학은 필연적으로 풍부한 Python 생태계(OpenCV, pandas, NumPy 등)와 웹 기반의 통합 개발 환경을 기본으로 제공하는 형태로 귀결됩니다.[2]

이러한 접근 방식은 개발 초기 단계의 편의성과 속도를 비약적으로 향상시키지만, 그 대가로 대용량의 루트 파일 시스템(rootfs)을 요구하게 됩니다. PYNQ 3.1의 ARM 아키텍처용 베이스 이미지인 pynq_rootfs.arm.v3.1은 Ubuntu 22.04(Jammy Jellyfish) 데스크톱 배포판을 기반으로 하며, 이로 인해 데스크톱 환경에 준하는 방대한 양의 시스템 라이브러리와 패키지를 기본적으로 포함하게 됩니다.[2] 결과적으로 PYNQ의 거대한 이미지 크기는 버그나 비효율적인 설계의 산물이 아니라, '개발 생산성'이라는 핵심 설계 철학을 구현하기 위한 의도적인 선택의 직접적인 결과입니다. 따라서 사용자가 추구하는 '초경량화' 목표는 PYNQ의 기본 패러다임과 근본적으로 상충되는 지점에서 출발하며, 표준 빌드 도구만으로는 해결하기 어려운 본질적인 문제를 내포하고 있습니다.

1.2 sdbuild 프로세스와 이미지 크기

PYNQ 이미지는 sdbuild라는 Makefile 기반의 빌드 스크립트를 통해 여러 단계(Stage)에 걸쳐 생성됩니다.[2] 이 빌드 프로세스는 효율성과 재현성을 위해 사전 빌드된 보드-비종속적(board-agnostic) rootfs, 즉

pynq_rootfs.arm.tar.gz 파일을 다운로드하여 시작하는 것을 표준 절차로 권장합니다.[1] 문제의 핵심은 이 베이스 rootfs 파일 자체가 이미 수 기가바이트에 달하는 대용량이라는 점입니다.

빌드 과정은 다음과 같이 진행됩니다:

- **Stage 1:** 부트 파티션 생성 및 PetaLinux를 이용한 부트로더(FSBL, U-Boot), 리눅스 커널 빌드.
- **Stage 2 & 3:** 다운로드한 대용량 rootfs의 압축을 해제하고, python_packages_jammy, jupyter와 같은 PYNQ의 핵심 패키지 그룹을 설치합니다. 이 패키지들은 서로 복잡한 의존성 관계를 맺고 있어 특정 요소만을 분리하기 어렵게 만듭니다.[2]
- **Stage 4:** 보드별(Pynq-Z1, Pynq-Z2 등) 특정 드라이버나 설정, 예제 노트북 등을 추가합니다.[9]

실제로 PYNQ 커뮤니티의 한 사용자는 표준 PYNQ 빌드 절차를 따랐을 때 생성된 rootfs의 크기가 약 6.3GB에 달했다고 보고했습니다.[2] 이는 사용자의 목표치인 200MB와 약 32배의 현격한 차이를 보여주며, 표준

sdbuild 흐름 내에서 경량화를 달성하는 것이 얼마나 어려운 과제인지를 명확히 시사합니다.

제2장: pynqremote 빌드 타겟의 심층 분석

2.1 PYNQ.remote의 개념과 목적

PYNQ.remote는 PYNQ 3.1 버전에서 베타 기능으로 도입된 아키텍처로, 리소스가 제한된 임베디드 보드의 부담을 줄이기 위해 고안되었습니다.[11] 이 접근법의 핵심은 Jupyter 노트북 서버, 컴파일러, 대용량 라이브러리와 같은 무거운 개발 환경 관련 구성 요소들을 호스트 PC에서 실행하고, 타겟 보드에는 원격 제어를 위한 최소한의 서버 데몬과 핵심

pynq 라이브러리만 남기는 것입니다.[12] 이론적으로 이 아키텍처는 타겟 보드에 탑재되는 이미지의 크기를 획기적으로 줄일 수 있는 강력한 잠재력을 가지고 있으며, 이는 사용자의 경량화 목표와 정확히 일치하는 방향입니다.

2.2 make pynqremote 명령어와 ZYNQ7020 지원 현황

PYNQ 공식 문서 중 Remote Image Build Guide는 pynqremote 이미지 빌드 방법으로 make pynqremote BOARDS=<board_name> 명령어를 소개하고 있습니다. 주목할 점은 이 문서에서 <board_name>의 예시로 Zynq UltraScale+ 기반의 ZCU104와 함께 ZYNQ7020 기반의 Pynq-Z2를 명시적으로 언급한다는 것입니다.[12] 이는 PYNQ 개발팀이 ZYNQ7020 아키텍처에 대한

pynqremote 빌드를 분명히 의도했으며, 빌드 스크립트에도 관련 로직이 포함되어 있을 가능성이 높음을 시사합니다.

그러나 별도의 공식 문서인 Current Status 페이지에서는 PYNQ 3.1의 PYNQ.remote 기능에 대해 공식적으로 검증된*(Validated)* 플랫폼으로 ZCU104와 AUP-ZU3만을 명시하고 있습니다.[14] 이 두 보드는 모두 Zynq UltraScale+ MPSoC (aarch64 아키텍처)를 기반으로 합니다. PYNQ-Z1/Z2의 기반인 ZYNQ7020 (arm 아키텍처)은 이 검증된 플랫폼 목록에서 빠져 있습니다.

이러한 공식 문서 간의 미묘한 불일치는 중요한 함의를 가집니다. 빌드 가이드의 예시는 ZYNQ7020에서의 pynqremote 빌드가 가능할 수 있음을 암시하지만, 검증된 플랫폼 목록의 부재는 이것이 공식적으로 테스트되거나 안정성이 보장된 경로가 아님을 나타냅니다. 즉, PYNQ 3.1 출시 시점에서 ZYNQ7020을 위한 pynqremote 기능은 '실험적' 또는 '미검증' 상태로

간주해야 합니다. 따라서 이 경로를 시도하는 개발자는 예상치 못한 빌드 오류나 런타임 문제에 직면할 가능성이 높으며, 공식적인 기술 지원을 받기 어려울 수 있습니다.

2.3 성공 사례의 부재

위 분석을 뒷받침하듯, PYNQ 공식 포럼, GitHub 이슈 트래커, 관련 기술 커뮤니티 등 제공된 모든 자료와 광범위한 추가 조사를 통틀어, PYNQ-Z1 또는 PYNQ-Z2 보드에서 PYNQ 3.1 버전의 pynqremote 빌드를 성공적으로 완료하고 그 결과(이미지 크기, 작동 여부 등)를 공유한 공개적인 사례는 단 한 건도 발견되지 않았습니다. 이는 해당 기능이 ZYNQ7020 플랫폼에서는 아직 안정화되지 않았거나 널리 사용되지 않고 있음을 강력하게 시사합니다.

제**3**장**: 200MB** 목표 달성 가능성 정량적 평가

3.1 PYNQ sdbuild 프레임워크의 구조적 한계

pynqremote 빌드 타겟이 존재함에도 불구하고 200MB 목표 달성이 어려운 근본적인 이유는 PYNQ sdbuild 프레임워크의 구조 자체에 있습니다. PYNQ 포럼의 한 사용자는 이미지 경량화를 위해 sdbuild의 마지막 단계인 Stage4를 제거하고, Python 패키지 의존성을 정의하는 requirements.txt 파일을 수정하는 등 다양한 시도를 했습니다. 그러나 Stage2와 Stage3 과정에서 Jupyter와 전체 Python 과학 컴퓨팅 생태계(OpenCV, pandas 등)가 핵심 의존성으로 묶여 있어 강제로 설치되었고, 결국 경량화에 실패했다고 보고했습니다.[2]

이는 sdbuild 시스템이 특정 패키지들을 선택적으로 제외하기 어려운, 긴밀하게 통합된(monolithic) 구조로 설계되었음을 보여줍니다. make pynqremote 명령어는 이미 수 GB에 달하는 베이스 rootfs 위에 적용되는 빌드 옵션일 가능성이 높습니다. 즉, 이 명령어는 Jupyter 서비스를 비활성화하고 원격 접속 데몬을 추가하는 역할을 할 수는 있지만, 이미 파일 시스템에 설치된 방대한 라이브러리 파일들을 대규모로 삭제하는 기능은 수행하지 못할 것으로 보입니다. 따라서 sdbuild를 사용하는 한, 최종 이미지 크기는 베이스 rootfs의 크기에서 크게 벗어나기 어렵고, 200MB 목표 달성은 거의 불가능에 가깝습니다.

3.2 최소 시스템의 구성 요소별 용량 추정

진정한 경량화를 위해서는 sdbuild를 벗어나 필수 구성 요소만으로 시스템을 재구성해야 합니다. ZYNQ7020 기반의 최소 PYNQ 실행 환경에 필요한 구성 요소와 예상 용량은 다음과 같습니다.

- 부트로더 **(BOOT.BIN):** FSBL(First Stage Boot Loader), U-Boot, 그리고 FPGA 비트스트림을 포함하며, 일반적으로 수 MB에서 최대 10MB 내외의 크기를 가집니다.
- 리눅스 커널 **(image.ub):** Zynq 플랫폼용으로 압축된 리눅스 커널 이미지(uImage)는 약 5MB에서 15MB 사이입니다.
- 최소 **Rootfs:** 모든 불필요한 유틸리티를 제거하고 BusyBox와 같은 최소한의 셸 환경으로 구성된 루트 파일 시스템은 10MB에서 20MB로 구축할 수 있습니다.
- **Python** 인터프리터 및 표준 라이브러리**:** 임베디드 환경을 위한 경량화된 Python 3 인터프리터와 필수 표준 라이브러리를 설치하는 데 약 30MB에서 50MB가 필요합니다.
- **pynq Python** 패키지 및 의존성**:** pynq 패키지의 소스 배포판(sdist) 크기만 해도 63MB에 달합니다.[15] 실제 설치 시에는 이보다 작아질 수 있지만, numpy와 같은 핵심 의존성 패키지를 포함하면 수십 MB의 공간이 추가로 필요합니다.

이들 핵심 요소의 용량을 보수적으로 합산하더라도 이미 100MB를 쉽게 초과할 수 있습니다. 여기에 시스템 운영에 필수적인 각종 공유 라이브러리(예: libc), 디바이스 드라이버, 네트워크 유틸리티 등을 추가하면 200MB는 매우 도전적인 목표 수치임을 알 수 있습니다.

제**4**장**:** 목표 달성을 위한 대안 전략 및 실행 계획

사용자의 최종 목표인 'ZYNQ7020에서 실행되는 경량 PYNQ 시스템'을 달성하기 위해서는 PYNQ의 표준 sdbuild 프레임워크를 우회하여, 처음부터 최소한의 구성 요소만을 선택적으로 쌓아 올리는 '상향식(Bottom-up)' 접근 방식이 필수적입니다. 이를 위한 가장 현실적이고 강력한 두 가지 전략은 AMD/Xilinx의 공식 임베디드 리눅스 개발 도구인 PetaLinux를 사용하거나, 초경량화에 특화된 Buildroot를 사용하는 것입니다.

아래 표는 각 방법론의 주요 특성을 비교하여 사용자가 자신의 프로젝트 요구사항에 가장 적합한 경로를 선택할 수 있도록 돕습니다.

표 **4.1: PYNQ** 이미지 생성 방법론 비교 분석

평가 기준	PYNQ sdbuild (표준/pynqremote)	PetaLinux + 수동 PYNQ 설치	Buildroot + 수동 PYNQ 설치
사용 편의성	높음 (단일 make	중간 (PetaLinux 워크플로우 학습	낮음 (Buildroot 구성에 대한 깊은

	명령어)	필요)	이해 필요)
예상 이미지 크기	큼 (수 GB) / 미검증 (수백 MB 이상 추정)	중간 (200-500MB 범위로 제어 가능)	매우 작음 (100MB 이하 가능)
커스터마이징 자유도	낮음 (정해진 프레임워크)	높음 (커널, rootfs 패키지 단위 제어)	매우 높음 (라이브러리 수준의 정밀 제어)
공식 지원 수준	PYNQ 프레임워크 자체는 공식 지원	PetaLinux는 AMD 공식 지원, PYNQ 통합은 사용자 책임	커뮤니티 기반 지원
권장 사용 사례	빠른 프로토타이핑, 연구/교육	리소스 제약이 있는 제품 개발 **(**사용자에게 최적**)**	극도의 경량화가 요구되는 특수 목적 시스템

4.1 전략 **A: PetaLinux** 기반의 미니멀 시스템 구축 **(**강력 권장**)**

이 전략은 AMD의 공식 도구를 사용하여 안정적인 기반을 확보하면서도, rootfs 구성을 통해 충분한 수준의 경량화를 달성할 수 있는 최적의 절충안입니다. 안정성과 커스터마이징 사이의 균형을 제공하여 사용자의 목표에 가장 부합하는 현실적인 경로입니다.

단계별 실행 계획**:**

1. **Vivado** 프로젝트 생성**:** Vivado Design Suite를 사용하여 PYNQ-Z1/Z2 보드에 맞는 ZYNQ7020 Processing System(PS)을 포함하는 최소한의 하드웨어 디자인을 생성합니다. 이후 File -> Export -> Export Hardware 메뉴를 통해 하드웨어 플랫폼 사양 파일(.xsa)을 생성합니다.
2. **PetaLinux** 프로젝트 생성**:** PetaLinux 개발 환경에서 다음 명령어를 실행하여 Zynq 템플릿 기반의 새 프로젝트를 생성합니다.[16]

   ```
   petalinux-create --type project --template zynq --name <project_name>
   ```
3. 하드웨어 구성 연동**:** 생성된 PetaLinux 프로젝트 디렉토리로 이동한 후, 1단계에서 생성한 .xsa 파일을 지정하여 하드웨어 정보를 프로젝트에 반영합니다. 이 과정은 PetaLinux가 보드에 맞는 부트로더와 커널을 올바르게 구성하도록 합니다.[17]

petalinux-config --get-hw-description=<path_to_xsa_directory>

4. **Rootfs** 최소화 및 **Python** 추가**:** 가장 핵심적인 단계로, petalinux-config -c rootfs 명령어를 실행하여 루트 파일 시스템 구성 메뉴에 진입합니다. 여기서 Filesystem Packages 메뉴로 이동하여 불필요한 패키지(웹 서버, GUI 라이브러리 등)를 모두 비활성화합니다. 그 다음, Filesystem Packages -> devel -> python3 메뉴에서 python3-core, python3-pip 등 PYNQ 실행에 필요한 최소한의 Python 패키지만을 선택하여 활성화합니다.[17]
5. 시스템 빌드**:** 구성이 완료되면 petalinux-build 명령어를 실행하여 부트로더, 커널, 그리고 최소화된 rootfs를 포함한 전체 시스템 이미지를 빌드합니다.[16]
6. 이미지 패키징 및 **SD**카드 준비**:** petalinux-package 명령어를 사용하여 부팅에 필요한 BOOT.BIN 파일을 생성합니다. 이후 빌드 결과물(BOOT.BIN, image.ub, rootfs.tar.gz)을 FAT32 및 ext4로 파티셔닝된 SD 카드에 올바르게 복사합니다.[18]
7. **pynq** 라이브러리 설치**:** 생성된 최소 리눅스 이미지로 보드를 부팅한 후, 이더넷을 통해 네트워크에 연결합니다. 마지막으로 pip3 install pynq 명령어를 실행하여 PYNQ 라이브러리와 그 의존성 패키지들을 설치합니다.

4.2 전략 **B: Buildroot**를 이용한 초경량 시스템 구축

Buildroot는 PetaLinux(Yocto 프로젝트 기반)보다 더 경량화에 특화된 빌드 시스템입니다. C 라이브러리(glibc, uClibc, musl) 선택부터 모든 패키지를 소스에서 직접 빌드하므로, 수십 MB 크기의 초경량 루트 파일 시스템 생성이 가능합니다.[22] 200MB 목표를 확실하게 달성하거나 그 이하로 줄여야 하는 극한의 상황에서는 가장 강력한 대안이 될 수 있습니다.

다만, Buildroot는 PetaLinux에 비해 Zynq 관련 설정(예: BOOT.BIN 생성)에 더 많은 수동 구성과 깊은 이해를 요구할 수 있습니다.[24] 실행 계획은 PetaLinux와 유사하게

make menuconfig를 통해 타겟 아키텍처, 툴체인, 최소 패키지(Python3, pip 포함)를 선택하고 이미지를 빌드한 후, 보드에서 pip로 pynq를 설치하는 절차를 따릅니다.

결론 및 최종 권고

결론 요약**:** 사용자가 요청한 'PYNQ 3.1 pynqremote 빌드를 통해 ZYNQ7020 보드용 200MB 이하 이미지를 생성한 성공 사례'는 현재 공개된 자료에서는 찾을 수 없습니다. 분석 결과, pynqremote 빌드는 ZYNQ7020 플랫폼에 대해 공식적으로 검증되지 않은 실험적 경로로 판단됩니다. 또한, PYNQ sdbuild 프레임워크는 생산성에 초점을 맞춘 구조적 특성상, 베이스

이미지의 크기가 커서 200MB라는 경량화 목표 달성이 현실적으로 어렵습니다.

최종 권고: 사용자의 근본적인 목표인 'ZYNQ7020에서 동작하는 경량 PYNQ 시스템'을 구축하기 위한 가장 신뢰성 있고 실용적인 방법은 전략 **A: PetaLinux** 기반의 미니멀 시스템 구축입니다. 이 방법은 AMD의 공식 개발 도구를 사용하여 시스템의 안정성을 확보하면서도, 루트 파일 시스템의 패키지를 정밀하게 제어하여 요구사항에 근접하는 수준의 경량화를 달성할 수 있는 최적의 절충안입니다.

향후 연구 제언: pynqremote 빌드가 ZYNQ7020에서 실패하는 구체적인 원인을 분석하여 PYNQ 커뮤니티에 기여하거나, Buildroot를 사용하여 100MB 이하의 초경량 PYNQ 이미지를 구현하는 것은 의미 있는 후속 연구가 될 수 있습니다. 또한, 사용자는 PYNQ 공식 포럼(discuss.pynq.io)을 통해 ZYNQ7020 플랫폼에 대한 pynqremote 기능의 공식 지원 로드맵을 문의하여 향후 업데이트 계획을 확인할 수 있습니다.

참고 자료

1. Pynq 2.7 for Zybo-Z7 - Learn, 9월 17, 2025에 액세스, https://discuss.pynq.io/t/pynq-2-7-for-zybo-z7/4124
2. How to build a minimal PYNQ 3.1 image (no Jupyter), only pynq Python library for Overlay(), fitting in 8 GB eMMC, 9월 17, 2025에 액세스, https://discuss.pynq.io/t/how-to-build-a-minimal-pynq-3-1-image-no-jupyter-only-pynq-python-library-for-overlay-fitting-in-8-gb-emmc/8712
3. PYNQ | Python Productivity for AMD Adaptive Computing platforms, 9월 17, 2025에 액세스, http://www.pynq.io/
4. Python productivity for Zynq (Pynq) Documentation - Read the Docs, 9월 17, 2025에 액세스, https://buildmedia.readthedocs.org/media/pdf/pynq/v2.4/pynq.pdf
5. Xilinx/PYNQ: Python Productivity for ZYNQ - GitHub, 9월 17, 2025에 액세스, https://github.com/Xilinx/PYNQ
6. PYNQ supported boards and PYNQ pre-built images | PYNQ, 9월 17, 2025에 액세스, http://www.pynq.io/boards.html
7. PYNQ SD Card image — Python productivity for Zynq (Pynq) - Read the Docs, 9월 17, 2025에 액세스, https://pynq.readthedocs.io/en/latest/pynq_sd_card.html
8. sdbuild · image_v2.6.0 · paulrr2 / PYNQ - GitLab at Illinois, 9월 17, 2025에 액세스, https://gitlab-beta.engr.illinois.edu/paulrr2/PYNQ/-/tree/image_v2.6.0/sdbuild
9. PYNQ Edition! Building PYNQ Images - Hackster.io, 9월 17, 2025에 액세스, https://www.hackster.io/news/pynq-edition-building-pynq-images-4f0ec30e172f
10. PYNQ SD Card image — Python productivity for Zynq (Pynq) - Read the Docs, 9월 17, 2025에 액세스, https://pynq.readthedocs.io/en/v2.7.0/pynq_sd_card.html
11. Releases · Xilinx/PYNQ - GitHub, 9월 17, 2025에 액세스, https://github.com/Xilinx/PYNQ/releases
12. Remote Image Build Guide — Python productivity for Zynq (Pynq) - Read the Docs, 9월 17, 2025에 액세스, https://pynq.readthedocs.io/en/v3.1/pynq_remote/image_build.html
13. PYNQ Introduction — Python productivity for Zynq (Pynq), 9월 17, 2025에 액세스,

https://pynq.readthedocs.io/
14. Current Status — Python productivity for Zynq (Pynq) - Read the Docs, 9월 17, 2025에 액세스, https://pynq.readthedocs.io/en/v3.1/pynq_remote/status.html
15. pynq - PyPI, 9월 17, 2025에 액세스, https://pypi.org/project/pynq/
16. PetaLinux Tutorial+Demo - CERN Indico, 9월 17, 2025에 액세스, https://indico.cern.ch/event/799275/contributions/3413736/attachments/1861349/3059191/Petalinux_TutorialDemo.pdf
17. Running Petalinux on Zynq SoC from scratch - Zybo board, 9월 17, 2025에 액세스, https://nuclearrambo.com/wordpress/running-petalinux-on-zynq-soc-from-scratch-zybo-board/
18. Building and Debugging Linux Applications for Zynq-7000 SoCs - GitHub Pages, 9월 17, 2025에 액세스, https://xilinx.github.io/Embedded-Design-Tutorials/docs/2021.1/build/html/docs/Introduction/Zynq7000-EDT/4-linux-for-zynq.html
19. Exploring Python on Zynq UltraScale - Adaptive Support - AMD, 9월 17, 2025에 액세스, https://adaptivesupport.amd.com/s/article/1171977
20. Path to Programmable 3 - Training Blog 3 - Petalinux Overview, customization for custom hardware design and "hello ultra96v2" from python - element14 Community, 9월 17, 2025에 액세스, https://community.element14.com/challenges-projects/design-challenges/pathprogrammable3/b/blog/posts/path-to-programmable-3---training-blog-3---petalinux-overview-customization-for-custom-hardware-design-and-hello-ultra96v2-from-python
21. Developing Custom Petalinux for Xilinx Zynq Ultrascale+ MPSoc | by Shubham Chhetri, 9월 17, 2025에 액세스, https://medium.com/@shubhamchh3/1-prerequisites-6156c220bb89
22. Creating a root file system with Buildroot | DAVE Embedded Systems, 9월 17, 2025에 액세스, https://www.dave.eu/en/case-histories/creating-a-root-file-system-with-buildroot
23. Build Linux for Zynq-7000 AP SoC using Buildroot - Xilinx Wiki, 9월 17, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842369/Build+Linux+for+Zynq-7000+AP+SoC+using+Buildroot
24. buildroot patches for zynq platforms - GitHub, 9월 17, 2025에 액세스, https://github.com/leaflabs/buildroot-zynq
25. How to adapt buildroot for a custom Zynq FPGA bitstream - Stack Overflow, 9월 17, 2025에 액세스, https://stackoverflow.com/questions/49657157/how-to-adapt-buildroot-for-a-custom-zynq-fpga-bitstream