실리콘에서 **Python**까지**: PYNQ** 소프트웨어 스택과 커스텀 보드 구현을 위한 최종 가이드

Part I: PYNQ 환경 해부

이 보고서의 첫 번째 부분에서는 PYNQ-Z2 보드를 기준으로 PYNQ 소프트웨어 스택을 체계적으로 분석합니다. 하드웨어의 가장 낮은 수준에서 시작하여 각 추상화 계층을 거쳐 사용자 애플리케이션 단계까지 올라가며, 각 구성 요소의 역할과 PYNQ 생태계가 작동하는 데 있어 그 중요성을 심층적으로 탐구합니다. 이를 통해 PYNQ 프레임워크에 대한 근본적인 이해를 구축하는 것을 목표로 합니다.

Section 1: Zynq 아키텍처와 **PYNQ** 패러다임

이 섹션에서는 PYNQ를 이해하는 데 필수적인 하드웨어 및 소프트웨어의 기본 개념을 정립합니다. Zynq SoC의 독특한 아키텍처를 설명하고, PYNQ 철학이 이 아키텍처를 어떻게 활용하여 하드웨어와 소프트웨어 개발 간의 간극을 메우는지 분석합니다.

1.1 처리 시스템**(PS)**과 프로그래머블 로직**(PL)**의 공생 관계

PYNQ 환경의 기반이 되는 하드웨어는 AMD-Xilinx의 Zynq-7000 SoC(System on Chip)입니다. Zynq 아키텍처의 핵심은 단일 칩 내에 두 개의 서로 다른 컴퓨팅 패러다임을 통합한 하이브리드 구조에 있습니다.[1]

- 처리 시스템 **(Processing System, PS):** 듀얼 코어 ARM Cortex-A9 프로세서를 중심으로 하는 전통적인 마이크로프로세서 시스템입니다. 메모리 컨트롤러, USB, 이더넷, UART, SPI와 같은 표준 주변장치 컨트롤러를 포함하며, 소프트웨어 코드를 순차적으로 실행하는 데 최적화되어 있습니다.[1]

- 프로그래머블 로직 **(Programmable Logic, PL):** FPGA(Field-Programmable Gate Array) 패브릭으로, 하드웨어 회로를 사용자의 필요에 맞게 재구성할 수 있는 유연한 영역입니다. 병렬 처리가 요구되는 고성능 연산, 실시간 신호 처리, 맞춤형 I/O 인터페이스 구현 등에 탁월한 성능을 보입니다.[1]

이 두 시스템은 칩 내부의 고대역폭 AXI(Advanced eXtensible Interface) 상호연결 버스를 통해 긴밀하게 결합됩니다.[5] 이 AXI 인터페이스는 PS와 PL 간의 데이터 및 제어 신호를 전달하는 핵심적인 통로 역할을 하며, PYNQ 소프트웨어 스택이 구축되는 물리적 토대가 됩니다. 본 보고서의 기준 플랫폼인 PYNQ-Z2 보드는 Zynq XC7Z020 SoC, 512MB DDR3 메모리, 그리고 HDMI, 이더넷, 오디오 코덱과 같은 주요 주변장치를 탑재하고 있으며, 소프트웨어 스택은 이러한 하드웨어 자원을 완벽하게 제어하고 관리해야 합니다.[9]

1.2 PYNQ의 핵심 철학: 소프트웨어 개발자를 위한 병렬 처리 추상화

PYNQ(Python Productivity for Zynq)는 AMD-Xilinx의 적응형 컴퓨팅 플랫폼을 더 쉽게 사용할 수 있도록 설계된 오픈 소스 프레임워크입니다.[12] PYNQ의 근본적인 목표는 소프트웨어 개발자, 시스템 아키텍트, 심지어 하드웨어 설계자까지도 Verilog나 VHDL과 같은 전통적인 하드웨어 기술 언어(HDL)를 사용하지 않고 Python을 통해 프로그래머블 로직의 강력한 성능을 활용할 수 있게 하는 것입니다.[1]

이러한 목표를 달성하기 위한 핵심 개념이 바로 '하드웨어 라이브러리', 즉 '오버레이(Overlay)'입니다.[4] 오버레이는 복잡한 PL 설계를 재사용 가능한 Python 객체로 추상화한 것으로, 소프트웨어 개발자가 소프트웨어 라이브러리를 import하여 사용하는 것과 유사한 방식으로 하드웨어 가속 기능을 사용할 수 있게 합니다.[1] 이 패러다임의 전환은 PYNQ를 단순한 도구가 아닌 생산성 프레임워크로 만드는 핵심 요소입니다.

이러한 접근 방식은 하드웨어 설계의 필요성을 없애는 것이 아니라, 하드웨어 설계자의 역할과 애플리케이션 개발자의 역할을 명확히 분리하는 데 그 의의가 있습니다. 하드웨어 전문가는 재사용 가능하고 설정 가능한 하드웨어 모듈(오버레이)을 제작하고, 소프트웨어 전문가는 이 모듈을 고수준 Python API를 통해 소비합니다. 이는 하드웨어 개발을 현대적인 API 기반 소프트웨어 개발 방식과 유사하게 만들어, 하드웨어 자산의 재사용성과 접근성을 극대화하는 효과를 낳습니다.

Table 1: PYNQ 소프트웨어 스택 개요

계층 (Layer)	주요 구성 요소 (Key Components)	주 기능 (Primary Function)	하위 계층과의 관계
사용자 애플리케이션	Jupyter Notebooks, Python 스크립트	대화형 개발, 데이터 시각화, 애플리케이션 로직 구현	PYNQ 프레임워크의 Python API를 호출하여 하드웨어 제어
PYNQ 프레임워크	pynq Python 라이브러리, 오버레이 (.bit, .hwh)	하드웨어 기능 추상화, PL 동적 재구성, PS-PL 통신 API 제공	커널 드라이버를 사용하여 PL의 IP와 데이터를 주고받음
운영체제/커널	Ubuntu 기반 Rootfs, Linux 커널	프로세스 스케줄링, 메모리 관리, 하드웨어 자원 접근 제어	부트로더에 의해 메모리에 로드되며, 하드웨어 자원을 관리
부트로더	U-Boot, FSBL (First-Stage Boot Loader)	하드웨어 초기화, 커널 및 필수 파일 시스템 메모리 로딩	하드웨어의 부팅 모드에 따라 실행되며, OS 커널에 제어권 전달
하드웨어	Zynq SoC (PS + PL), DDR 메모리, 주변장치	물리적 연산 및 데이터 저장 수행	전원이 인가되면 부트로더 코드를 실행

Section 2: PYNQ 시스템의 탄생**:** 부트로더와 커널

이 섹션에서는 PYNQ-Z2 보드가 전원이 꺼진 상태에서 완전한 기능을 갖춘 Linux 시스템으로 부팅되는 과정을 상세히 설명합니다. 이는 소프트웨어 스택의 가장 낮은 수준에 해당하며, 특히 커스텀 보드 개발에 있어 기초가 되는 핵심적인 부분입니다.

2.1 부팅 시퀀스**:** 전원 인가부터 **init** 프로세스까지

Zynq SoC의 부팅 과정은 여러 단계의 로더를 거치는 계층적 구조를 가집니다. 이 과정은 하드웨어를 사용할 수 있는 상태로 만들고, 운영체제를 메모리에 올려 실행시키는 일련의 절차입니다.

1. **BootROM**과 **FSBL (First-Stage Boot Loader):** 전원이 인가되면 Zynq PS의 ARM 코어 중 하나가 칩 내부에 고정된 BootROM 코드를 실행합니다.[17] BootROM의 유일한 임무는 SD 카드나 QSPI 플래시와 같은 비휘발성 저장 매체에서 FSBL을 찾아 메모리로 로드하는 것입니다.[17] FSBL은 보드에 특화된 작은 프로그램으로, MIO 핀 설정, DDR 메모리 컨트롤러 초기화, 클럭 PLL 설정과 같은 필수적인 하드웨어 초기화를 수행합니다.[17] FSBL의 마지막 임무는 다음 단계의 부트로더인 U-Boot를 DDR 메모리에 로드하는 것입니다.
2. **U-Boot (The Universal Boot Loader):** 2단계 부트로더(SSBL)인 U-Boot는 FSBL보다 훨씬 강력하고 유연한 환경을 제공합니다.[18] U-Boot는 Linux 커널 이미지, 장치 트리, 그리고 루트 파일 시스템을 메모리에 로드하는 역할을 담당합니다. 또한, 네트워크(TFTP)를 통한 부팅이나 디버깅을 위한 대화형 셸을 제공하며, bootargs와 같은 커널 부팅 파라미터를 설정하는 중요한 역할을 수행합니다.[20]
3. 커널 로딩과 장치 트리 블롭 **(Device Tree Blob, DTB):** U-Boot가 부팅 과정에서 수행하는 마지막 작업은 제어권을 Linux 커널(uImage 또는 Image 파일)에 넘기는 것입니다.[18] 이때, 커널에 장치 트리 블롭(.dtb 파일)의 메모리 주소를 함께 전달합니다. DTB는 보드에 존재하는 하드웨어 구성(메모리 주소, 주변장치, 인터럽트 등)을 표준화된 형식으로 기술하는 데이터 구조입니다. 이를 통해 동일한 커널 바이너리가 서로 다른 하드웨어 구성을 가진 여러 보드를 지원할 수 있게 됩니다.[18]

2.2 운영체제 기반: Ubuntu 기반 임베디드 Linux

PYNQ는 최소한의 기능만을 포함하는 일반적인 임베디드 시스템과 달리, 완전한 기능을 갖춘 Ubuntu 기반의 Linux 배포판을 사용한다는 점에서 차별화됩니다.[24] 이는 시스템의 최소 용량보다는 개발자의 생산성과

apt-get과 같은 패키지 관리자를 통한 방대한 소프트웨어 생태계 접근성을 우선시하는 설계 철학을 반영합니다.[25]

- **Linux** 커널의 역할**:** 커널은 운영체제의 핵심으로, Zynq SoC의 모든 하드웨어 자원을 관리합니다. ARM 코어에서의 프로세스 스케줄링, 가상 메모리를 포함한 메모리 관리, 그리고 장치 드라이버를 통해 하드웨어에 대한 일관된 인터페이스를 제공하는 역할을 합니다.[19]
- 루트 파일 시스템 **(rootfs):** 이는 /bin, /etc, /lib 등 운영체제가 기능하는 데 필요한 모든 시스템 라이브러리, 유틸리티, 설정 파일들을 포함하는 디렉터리 구조입니다.[19]

PYNQ에서는 이 루트 파일 시스템이 SD 카드에 저장되며, Ubuntu 사용자 공간 전체와 Python, Jupyter, 그리고 pynq 라이브러리를 포함합니다. 커널은 부팅 과정의 마지막 단계에서 이 루트 파일 시스템을 마운트하고, 시스템의 첫 번째 사용자 프로세스인 init(PID 1)을 실행하여 나머지 모든 사용자 공간 서비스를 시작시킵니다.[19]

PYNQ가 최소화된 빌드 시스템 대신 Ubuntu를 채택한 것은 생산성이라는 핵심 목표를 직접적으로 지원하기 위한 전략적 결정입니다. 일반적인 임베디드 시스템은 Yocto나 Buildroot와 같은 도구를 사용하여 시스템의 크기와 자원 사용량을 최소화하는 데 집중합니다.[29] 반면, PYNQ는

apt-get과 같은 패키지 관리자를 기본으로 제공하여 개발자가 필요한 라이브러리를 전체 이미지를 재빌드할 필요 없이 즉시 설치할 수 있게 합니다. 이는 개발 과정에서 엄청난 생산성 향상을 가져옵니다. 커스텀 보드를 개발할 때, 이 점은 중요한 시사점을 제공합니다. 개발 플랫폼의 목표가 제한된 자원 환경의 최종 제품인지, 아니면 유연한 개발 환경인지에 따라 OS를 최소한으로 구성할지, 아니면 PYNQ처럼 풍부한 기능을 제공할지를 결정해야 합니다.

Section 3: 핵심 인터페이스**: PS-PL** 통신을 위한 **Linux** 드라이버

이 섹션에서는 Linux 커널 내부에 존재하며, 고수준 운영체제와 PL의 커스텀 하드웨어 간의 직접적인 중재자 역할을 하는 소프트웨어 계층을 탐구합니다. 이 드라이버들을 이해하는 것은 PYNQ가 어떻게 하드웨어를 제어하는지를 파악하는 데 핵심적입니다.

3.1 PL 프로그래밍**: FPGA Manager** 프레임워크

Linux가 실행되는 동안 오버레이의 .bit 파일을 PL에 로드하기 위해, PYNQ는 표준 Linux FPGA Manager 프레임워크를 사용합니다.[31] 이 프레임워크는 FPGA 프로그래밍을 위한 일반적인 API를 제공하며, Zynq에 특화된 구형

xdevcfg 드라이버를 대체하였습니다.[31]

pynq 라이브러리는 이 프레임워크와 상호작용하여 동적으로 PL을 재구성합니다.

3.2 저수준 데이터 및 제어 채널

PYNQ는 PS와 PL 간의 통신을 위해 여러 표준 Linux 커널 메커니즘을 활용합니다.

- 메모리 맵 **I/O (MMIO)**를 통한 레지스터 접근**:** PL에 구현된 커스텀 IP는 주로 AXI-Lite 슬레이브 인터페이스에 연결된 메모리 맵 레지스터를 통해 제어됩니다. ARM 프로세서 관점에서 이 레지스터들은 시스템 메모리 맵의 특정 주소로 보입니다. Linux 커널은 /dev/mem 장치 파일이나 UIO(Userspace I/O) 드라이버와 같은 메커니즘을 제공하여 사용자 공간 프로그램이 이러한 물리 메모리 주소에 직접 접근할 수 있도록 합니다.[8] pynq.MMIO 클래스는 이러한 저수준 메커니즘에 대한 Python 래퍼(wrapper)입니다.
- **AXI DMA**를 통한 고속 데이터 전송**:** 비디오 프레임이나 대용량 신호 처리 샘플과 같은 큰 데이터 블록을 PL과 PS의 DDR 메모리 간에 전송할 때 MMIO는 속도가 너무 느립니다. 해결책은 DMA(Direct Memory Access)입니다. PL의 AXI DMA IP 코어는 CPU의 개입 없이 직접 DDR 메모리를 읽거나 쓸 수 있습니다. PYNQ는 AXI DMA 엔진을 관리하기 위해 상응하는 커널 드라이버를 사용합니다.[6]
- 인터럽트를 통한 하드웨어 이벤트 처리**:** PL은 인터럽트 라인을 사용하여 PS에 특정 이벤트가 발생했음을 알릴 수 있습니다. PYNQ는 이러한 인터럽트들이 PL의 AXI Interrupt Controller를 통해 라우팅된 후, PS의 주 인터럽트 입력(IRQ_F2P)에 연결될 것을 요구합니다.[7] PYNQ는 커널의 UIO 프레임워크(특히 uio_pdrv_genirq)를 사용하여 이러한 인터럽트를 사용자 공간 애플리케이션에 노출시키고, pynq.Interrupt 클래스가 이를 활용합니다.[24]
- 물리적 연속 메모리 관리 **(cma):** DMA 엔진은 가상 메모리가 아닌 물리적 메모리에서 연속적인 버퍼를 필요로 합니다. 표준 Linux 메모리 할당자는 이를 보장하지 않습니다. PYNQ는 커널의 CMA(Contiguous Memory Allocator) 프레임워크에 의존합니다. 레거시 xlnk 드라이버가 초기에 이 인터페이스 역할을 했지만, 현대의 PYNQ는 CMA와 상호작용하여 이러한 특수 버퍼를 할당하는 Python 래퍼를 사용합니다.[7] pynq.allocate 함수가 이를 위한 고수준 API입니다.

PYNQ의 고수준 API는 사실상 사용자 공간 하드웨어 상호작용을 위한 표준적이고 잘 정립된 Linux 커널 인터페이스(UIO, CMA, FPGA Manager 등) 위에 구축된 얇지만 강력한 Python 래퍼입니다. 이는 PYNQ 개발자들이 완전히 새로운 커널 인프라를 만드는 대신, 기존의 안정적인 커널 기능을 활용하는 전략적 선택을 했음을 보여줍니다. PYNQ의 핵심 기술은 커널 자체가 아니라, 강력하지만 때로는 복잡한 커널 인터페이스를 사용하기 쉽게 만드는 사용자 공간 Python 라이브러리에 있습니다. 커스텀 보드 개발자에게 이는 커널 수준의 주된 작업이 새로운 커널 코드를 생성하는 것이 아니라, 이러한 표준 드라이버들을 활성화하는 설정 작업임을 의미합니다.

Table 2: PS-PL 통신 방식 매핑

통신 작업	하드웨어 인터페이스 (PL)	커널 드라이버/프레임워크	pynq API 클래스
레지스터 읽기/쓰기	AXI-Lite Slave	/dev/mem 또는 UIO	pynq.MMIO
대용량 데이터 전송	AXI-Stream w/ AXI DMA	AXI DMA Driver	pynq.DMA
이벤트 신호 처리	Interrupt Pin w/ AXI Intc	UIO (uio_pdrv_genirq)	pynq.Interrupt
공유 메모리 버퍼	AXI-Full Master	CMA (Contiguous Memory Allocator)	pynq.allocate

---

Section 4: PYNQ의 심장**: Python** 프레임워크와 오버레이 추상화

이 섹션에서는 사용자 공간의 pynq Python 라이브러리를 집중적으로 분석하며, 프레임워크를 강력하게 만드는 핵심 추상화 개념인 오버레이에 대해 심도 있게 다룹니다.

4.1 pynq Python 라이브러리 아키텍처

pynq 패키지는 PL과 상호작용하기 위한 API를 제공하는 Python 모듈의 집합입니다.[38] 이 패키지는

pip를 통해 설치할 수 있으며, 일반적인 IP를 위한 고수준 드라이버(예: pynq.lib.axigpio, pynq.lib.dma)와 직접적인 하드웨어 조작을 위한 저수준 인터페이스(예: pynq.MMIO, pynq.allocate)를 모두 포함하고 있습니다.[7] 또한, MicroBlaze와 같은 소프트 프로세서를 지원하여 실시간 작업을 주 ARM 프로세서에서 오프로드할 수 있는 기능도 제공합니다.[40]

4.2 오버레이**:** 하드웨어 라이브러리 심층 분석

오버레이는 PYNQ에서 하드웨어 기능의 기본 단위입니다. 이는 단순한 비트스트림이 아니라, 최소 두 개의 핵심 파일로 구성된 패키지입니다.[5]

- 비트스트림 **(.bit 파일):** Vivado에서 생성된 이진 데이터로, PL 내부의 로직 게이트와 상호연결을 물리적으로 구성합니다.[5]
- 하드웨어 핸드오프 **(.hwh 파일):** Vivado에서 비트스트림과 함께 생성되는 XML 기반의 메타데이터 파일입니다. 이 파일은 PYNQ 프레임워크의 동적 특성을 가능하게 하는 가장 중요한 요소입니다. PL 내부의 모든 IP 블록, 그들의 유형(VLNV), 메모리 맵 주소 범위, 인터럽트 연결, GPIO 신호 등 전체 하드웨어 시스템을 기술합니다.[41]

사용자가 Python 코드에서 ol = Overlay('my_design.bit')를 실행하면, pynq 라이브러리는 내부적으로 다음과 같은 일련의 작업을 수행합니다 [42]:

1. 해당하는 my_design.hwh 파일을 찾습니다.
2. FPGA Manager 드라이버를 사용하여 .bit 파일을 PL에 다운로드합니다.
3. .hwh 파일을 파싱하여 하드웨어 설계에 대한 메모리 내 딕셔너리(ip_dict)를 구축합니다.[44]
4. Vivado 블록 다이어그램에 있는 IP 이름에 해당하는 Python 속성을 ol 객체에 동적으로 생성합니다.
5. .hwh 파일에서 확인된 IP 유형에 따라 각 속성에 적절한 Python 드라이버를 바인딩합니다.

4.3 하드웨어 **IP**의 **Pythonic** 제어

- **DefaultIP** 드라이버**:** .hwh 파서가 특정 드라이버가 등록되지 않은 IP 블록을 발견하면, 일반적인 DefaultIP 드라이버를 바인딩합니다.[47] 이 강력한 기능은 모든 커스텀 IP에 대한 즉각적인 저수준 접근을 제공합니다. 개발자는 my_ip.read(offset) 및 my_ip.write(offset, data)를 사용하여 IP의 레지스터와 상호작용할 수 있으며, 이를 통해 Python에서 직접 신속한 프로토타이핑과 디버깅이 가능해집니다.[48]
- 커스텀 고수준 **Python** 드라이버 제작 방법론**:** 더 사용자 친화적인 API를 제공하기 위해 개발자는 자신만의 드라이버를 작성할 수 있습니다. 이는 pynq.DefaultIP를 상속하고, 지원하는 IP 유형(예: 'xilinx.com:hls:add:1.0')을 나열하는 bindto 클래스 속성을 포함하는 Python 클래스를 만드는 방식으로 이루어집니다.[47] PYNQ 프레임워크는 이러한 커스텀 드라이버를 자동으로 발견하고 등록하여, 오버레이가 로드될 때 DefaultIP 대신 해당 드라이버를 바인딩합니다.[49] 이를 통해 레지스터 오프셋과 같은 저수준 세부 사항을 my_ip.add(5, 10)과 같이 의미 있는 메서드로 추상화할 수 있습니다.[50]

.hwh 파일과 DefaultIP 드라이버의 조합은 PYNQ 생산성의 핵심인 강력한 "발견 및 디버그" 루프를 형성합니다. 하드웨어 엔지니어는 Vivado에서 새 설계를 생성하고 .bit와 .hwh 파일을 PYNQ 보드로 복사한 후, 즉시 Overlay()를 통해 로드할 수 있습니다. .hwh 파서는 새로운 IP를 자동으로 발견하고 DefaultIP 드라이버를 바인딩하여 Jupyter 노트북에서 레지스터에 대한 즉각적인 읽기/쓰기 접근을 제공합니다. 이는 하드웨어 합성이 완료된 후 불과 몇 분 만에 실제

Linux/Python 환경에서 IP를 테스트할 수 있게 하여, 하드웨어/소프트웨어 공동 설계 과정을 극적으로 가속화합니다. 이는 매번 사소한 하드웨어 변경마다 C 테스트 애플리케이션 작성, 크로스 컴파일, 배포 과정을 거쳐야 했던 전통적인 임베디드 워크플로우와는 근본적으로 다른 접근 방식입니다.

Section 5: 대화형 환경: 엣지에서의 Jupyter

Part I의 마지막 섹션에서는 스택의 최상위 계층인 사용자 대면 Jupyter Notebook 환경을 살펴봅니다.

5.1 시스템 아키텍처: Zynq PS에서 Jupyter 호스팅

PYNQ 환경은 Zynq의 ARM 프로세서에서 직접 웹 서버를 실행합니다.[1] 이 서버는 Jupyter Notebook 또는 JupyterLab 애플리케이션을 호스팅합니다. 사용자는 이더넷을 통해 보드에 연결하고 표준 웹 브라우저를 통해 시스템과 상호작용합니다.[2] 보드의 IP 주소(예:

http://192.168.2.99:9090)가 진입점이 됩니다.[2]

이 구조에서 브라우저는 씬 클라이언트(thin client) 역할을 하며, 실제 코드 실행은 Zynq PS에서 실행되는 IPython 커널 내에서 이루어집니다.[1] 이러한 아키텍처는 사용자 관점에서 PYNQ를 플랫폼에 구애받지 않게 만들어 줍니다. 필요한 것은 오직 웹 브라우저뿐입니다.

5.2 Python 과학 스택을 통한 가속 컴퓨팅 활성화

PYNQ 이미지는 NumPy, Matplotlib, OpenCV와 같이 과학 컴퓨팅, 데이터 분석, 시각화를 위한 표준 Python 라이브러리 생태계가 사전 설치된 상태로 제공됩니다.[37]

이를 통해 표준 Python 라이브러리를 사용하여 데이터를 준비하고, 오버레이를 통해 PL의 하드웨어 가속 함수로 전달한 다음, 그 결과를 다시 동일한 Python 라이브러리를 사용하여 분석하거나 시각화하는 원활한 워크플로우가 가능해집니다. 이 모든 과정이 단일 대화형 Jupyter 노트북 내에서 이루어집니다.[48] 소프트웨어 라이브러리와 하드웨어 오버레이의 이러한 긴밀한 통합이 바로 PYNQ 개발 경험의 본질입니다.

Jupyter를 채택한 것은 임의의 결정이 아닙니다. Jupyter의 "문서화 컴퓨팅(literate computing)"

모델은 하드웨어/소프트웨어 공동 설계의 탐색적이고 문서화가 중요한 특성에 완벽하게 부합합니다. Jupyter 노트북은 실행 가능한 코드, 설명 텍스트(Markdown), 수식, 이미지, 플롯을 단일 문서에 결합할 수 있습니다.[13] 하드웨어 가속 시스템을 개발할 때, 개발자는 하드웨어 아키텍처를 문서화하고, 소프트웨어 API를 설명하며, 코드 예제를 제공하고, 성능 플롯과 같은 결과를 보여줘야 합니다. 전통적인 워크플로우에서는 이 정보가 설계 문서, 소스 코드 주석, 별도의 테스트 스크립트에 흩어져 있었을 것입니다. Jupyter 노트북은 이 모든 것을 실행 가능하고 공유 가능한 단일 문서로 통합하여, PYNQ를 개발뿐만 아니라 교육 및 재현 가능한 연구를 위한 훌륭한 도구로 만들어 줍니다. 노트북 자체가 오버레이에 대한 포괄적인 문서가 되는 것입니다.

Part II: 커스텀 보드 구현을 위한 실무 가이드

이 보고서의 두 번째 부분에서는 이론에서 실습으로 전환합니다. Part I에서 구축한 기초 지식을 활용하여, 커스텀 Zynq-7000 보드에 PYNQ 환경을 복제하기 위한 단계별 가이드를 제공합니다.

Table 3: 커스텀 보드 구현 워크플로우

단계 (Phase)	도구 (Tool)	입력 산출물 (Input Artifacts)	주요 작업/명령어 (Key Actions/Commands)	출력 산출물 (Output Artifacts)
1: 하드웨어 정의	Vivado	보드 회로도/사양	create_bd, export_hardware	하드웨어 플랫폼 파일 (.xsa)
2: OS 생성	PetaLinux	커스텀 XSA	petalinux-create, petalinux-config, petalinux-build,	보드 지원 패키지 (.bsp)

			petalinux-package --bsp	
3: PYNQ 이미지 조립	PYNQ sdbuild Makefile	커스텀 PetaLinux BSP, 커스텀 보드 .spec 파일	make BOARDS=...	부팅 가능한 SD 카드 이미지 (.img)

Section 6: Phase 1 - Vivado를 이용한 하드웨어 플랫폼 생성

이 섹션에서는 첫 번째 실질적인 단계인 Vivado에서 커스텀 하드웨어를 정의하고 소프트웨어 도구에 필요한 정보를 내보내는 과정을 상세히 설명합니다.

6.1 커스텀 보드 사양에 맞는 **Zynq PS** 설정

PYNQ 환경을 커스텀 보드에 포팅하는 첫 단계는 해당 보드의 하드웨어 사양을 Vivado에 정확하게 반영하는 것입니다.

1. Vivado에서 커스텀 보드에 탑재된 특정 Zynq-7000 부품을 대상으로 새 프로젝트를 생성합니다.[56]
2. 새 블록 다이어그램에 'Zynq7 Processing System' IP 블록을 추가합니다. 이 블록을 더블 클릭하여 재설정 GUI를 엽니다.
3. 이 GUI에서 커스텀 보드의 회로도와 일치하도록 모든 PS 설정을 구성해야 합니다. 여기에는 DDR 메모리 구성(메모리 유형, 타이밍, 버스 폭), 주변장치(UART, SDIO, 이더넷, USB)를 위한 MIO 핀 할당, 그리고 PS-PL 클럭 주파수 설정이 포함됩니다.[1] 만약 보드 제조사에서 제공하는 프리셋(preset) 파일이 있다면 이를 사용하는 것이 가장 정확하고 효율적인 방법입니다.[1]

6.2 최소한의 **PL** 기본 설계 생성

초기 PYNQ 포팅 단계에서는 PL 설계가 복잡할 필요가 없습니다. 목표는 합성이 가능한 유효한 설계를 만드는 것입니다. 최소한 PS의 FCLK_CLK0 클럭 출력을 마스터 AXI 포트 중 하나의 M_AXI_GP0_ACLK에 연결하여 PL에 클럭을 공급하고 AXI 인터커넥트를 활성화해야 합니다.

Vivado의 'Run Block Automation' 기능을 사용하면 이러한 기본 연결을 자동으로 적용할 수 있습니다.[56]

6.3 최종 하드웨어 플랫폼(XSA 파일) 생성 및 내보내기

1. 블록 다이어그램에 대한 HDL 래퍼를 생성하고, 합성(Synthesis), 구현(Implementation), 비트스트림 생성(Generate Bitstream) 단계를 순서대로 실행합니다.[56] PetaLinux와의 완벽한 호환성을 위해 비트스트림을 포함하는 것이 권장됩니다.[59]
2. Vivado 메뉴에서 File -> Export -> Export Hardware를 선택합니다.[59]
3. 출력물은 .xsa(Xilinx Support Archive) 확장자를 가진 단일 아카이브 파일입니다. 이 파일은 PS 설정, PL 비트스트림, 메모리 맵, IP 메타데이터 등 전체 하드웨어 정의를 캡슐화합니다. 이 XSA 파일은 다음 단계인 PetaLinux로 전달되는 매우 중요한 결과물입니다.[59]

Vivado에서 Zynq PS를 설정하는 것은 단순한 하드웨어 설계 단계를 넘어, 사실상 가장 중요하고 첫 번째인 소프트웨어 설정 단계입니다. PetaLinux는 XSA 파일 내의 정보를 기반으로 FSBL과 U-Boot 같은 부트 소프트웨어를 생성합니다. 만약 Vivado PS 설정에서 DDR 컨트롤러 설정이 잘못되면, PetaLinux가 생성한 FSBL은 RAM을 올바르게 초기화하지 못해 시스템 부팅에 실패할 것입니다. UART의 MIO 핀 설정이 틀리면 콘솔 출력을 전혀 볼 수 없게 됩니다. 따라서 Vivado의 Zynq PS 설정 GUI는 부트 소프트웨어의 가장 낮은 수준을 설정하는 고수준 설정 도구와 같습니다. 여기서의 실수는 단순한 하드웨어 버그가 아니라, 부팅 시 전체 시스템 장애로 나타나는 소프트웨어 버그가 됩니다. 이는 Zynq 플랫폼의 본질적인 "공동 설계(co-design)" 특성을 명확히 보여줍니다.

Section 7: Phase 2 - PetaLinux를 이용한 커스텀 Linux OS 빌드

하드웨어 정의가 완료되면, 이 섹션에서는 PetaLinux를 사용하여 커스텀 부트로더, 커널, 루트 파일 시스템을 빌드하는 방법을 설명합니다.

7.1 커스텀 XSA로부터 PetaLinux 프로젝트 생성

PetaLinux는 Xilinx 플랫폼을 위한 임베디드 Linux 개발을 간소화하기 위해 제공되는 Yocto Project 기반의 래퍼 도구입니다.[63]

1. 먼저, 일반 템플릿을 사용하여 새 PetaLinux 프로젝트를 생성합니다: petalinux-create

--type project --template zynq --name my_custom_board.[66]

2. 다음으로, 이전 단계에서 생성한 XSA 파일로부터 하드웨어 정의를 가져옵니다: petalinux-config --get-hw-description=<path_to_xsa_directory>.[60] 이 명령은 XSA 파일의 내용을 기반으로 프로젝트의 FSBL, U-Boot, 커널 장치 트리를 자동으로 구성합니다.

7.2 커널**, U-Boot,** 장치 트리 설정

petalinux-config 명령을 실행하여 최상위 시스템 설정 메뉴를 엽니다. 여기에서 주 부팅 장치(예: SD 카드)와 같은 시스템 수준 옵션을 설정할 수 있습니다. petalinux-config -c kernel 및 petalinux-config -c u-boot 명령을 사용하여 각각 커널과 U-Boot의 상세 설정 메뉴로 들어갈 수 있습니다. 이 메뉴에서 Section 3에서 논의된 특정 커널 드라이버(예: UIO, CMA, FPGA Manager)를 활성화해야 합니다.

7.3 루트 파일 시스템 커스터마이징**: Python, Jupyter, pynq** 패키지 통합

petalinux-config -c rootfs 명령을 실행하여 최종 루트 파일 시스템에 포함될 패키지를 설정합니다.[70] 메뉴를 탐색하여 Python 3 (

python3), pip, 그리고 필요한 라이브러리들을 활성화합니다.

Jupyter와 PYNQ는 기본 PetaLinux 레시피에 포함되어 있지 않으므로, 커스텀 Yocto 메타-레이어를 추가해야 합니다. meta-jupyter 레이어는 Jupyter와 그 의존성 패키지들의 레시피를 제공합니다.[71] 또한

pynq 패키지 자체에 대한 레시피를 직접 생성해야 하는데, 이는 Yocto 빌드 시스템에 pynq 소스 코드의 위치를 알려주고 빌드 및 설치 방법을 지시하는 것을 포함합니다.[72] 이는 고급 Yocto 작업이지만, 대규모 소프트웨어 프레임워크를 통합하는 올바른 방법입니다.

project-spec/meta-user/conf/petalinuxbsp.conf와 같은 설정 파일에 IMAGE_INSTALL_append = " python3-pynq python3-jupyter"와 같은 라인을 추가하여 필요한 패키지를 루트 파일 시스템에 포함시켜야 합니다.[73]

7.4 프로젝트 빌드 및 보드 지원 패키지**(BSP)** 패키징

1. petalinux-build 명령을 실행하여 FSBL, U-Boot, 커널, 루트 파일 시스템 등 모든 구성 요소를 컴파일합니다.[76] 이 과정은 상당한 시간이 소요될 수 있습니다.
2. 이 단계의 최종 목표는 전체 PetaLinux 프로젝트를 보드 지원 패키지(BSP) 파일로 패키징하는 것입니다: petalinux-package --bsp -o my_custom_board.bsp. 이 BSP 파일은 모든 PetaLinux 설정, 레시피, 사전 빌드된 구성 요소를 포함하는 단일 아카이브입니다. 이는 마지막 PYNQ 이미지 조립 단계의 핵심 입력물이 됩니다.[79]

PetaLinux는 하드웨어 인지 빌드 시스템 오케스트레이터로서의 역할을 수행합니다. 이 워크플로우에서 PetaLinux의 주 기능은 하드웨어 정의(XSA)를 일관된 소프트웨어 설정 집합(장치 트리, 부트로더 설정)으로 변환한 다음, Yocto 빌드 시스템을 사용하여 모든 것을 부팅 가능한 시스템으로 컴파일하는 것입니다. petalinux-config --get-hw-description 명령이 이 과정의 핵심입니다.[66] 이 명령은 XSA를 읽고, 이를 기반으로 Linux 커널에 어떤 주변장치가 어떤 주소에 활성화되어 있는지 알려주는 장치 트리를 생성하며, 메모리 맵을 인지하도록 U-Boot를 설정하고, PS 초기화 코드로 FSBL을 구성합니다. 즉, PetaLinux는 단순한 Linux 빌드 도구가 아니라 Xilinx 하드웨어를

이해하는 특화된 도구입니다. 이러한 자동화는 순수 Yocto를 사용하는 것에 비해 큰 이점을 제공하며, 커스텀 보드 개발자에게 하드웨어를 인지하는 부팅 가능한 Linux 시스템을 구축하는 가장 빠른 경로를 제시합니다.[63]

Section 8: Phase 3 - sdbuild를 이용한 최종 PYNQ 이미지 조립

마지막 섹션에서는 PYNQ 프로젝트의 자체 빌드 시스템인 sdbuild를 사용하여 커스텀 PetaLinux BSP로부터 최종적인 SD 카드 이미지를 생성하는 방법을 다룹니다.

8.1 PYNQ 빌드 시스템을 위한 커스텀 보드 저장소 구성

PYNQ sdbuild 흐름은 Makefile에 의해 구동되며 특정 디렉터리 구조를 요구합니다.[79]

1. 공식 PYNQ GitHub 저장소와 유사하게, 'boards' 디렉터리 내에 커스텀 보드를 위한 새 디렉터리(예: my_custom_board)를 생성합니다.[79]
2. 이 디렉터리 안에 이전 단계에서 생성한 커스텀 PetaLinux BSP 파일(my_custom_board.bsp)을 위치시킵니다.
3. 또한, 최소한의 Vivado 설계에서 생성된 .bit 및 .hwh 파일을 포함하는 base 오버레이 디렉터리를 생성합니다. 이것이 부팅 시 로드될 기본 오버레이가 됩니다.

8.2 보드 명세 **(.spec)** 파일 작성

보드 디렉터리 내에 my_custom_board.spec이라는 이름의 텍스트 파일을 생성합니다.[82] 이 파일은

sdbuild Makefile에 다음과 같은 핵심 정보를 제공합니다 [82]:

- ARCH_my_custom_board := arm (Zynq-7000의 경우)
- BSP_my_custom_board := my_custom_board.bsp
- BITSTREAM_my_custom_board := base/base.bit

이 파일은 추가로 설치할 패키지나 포함할 노트북을 명시하는 데에도 사용될 수 있습니다.[82]

8.3 sdbuild Makefile을 실행하여 부팅 가능한 SD 카드 이미지 생성

1. 공식 PYNQ GitHub 저장소를 복제합니다.
2. sdbuild 디렉터리로 이동합니다.
3. make 명령을 실행하여 빌드를 시작하되, 커스텀 보드 저장소 경로와 보드 이름을 지정합니다: make BOARDS=my_custom_board BOARDDIR=/path/to/my/boards/repo.[79]

sdbuild 스크립트는 다음과 같은 작업을 수행합니다:

1. 보드에 구애받지 않는 PYNQ 루트 파일 시스템(PYNQ 패키지가 포함된 사전 빌드된 Ubuntu)의 압축을 풉니다.
2. 커스텀 PetaLinux BSP를 추출합니다.
3. BSP의 구성 요소(부트로더, 커널 등)를 사용하여 부팅 파티션을 생성합니다.
4. BSP의 보드별 드라이버나 설정을 루트 파일 시스템에 통합합니다.
5. 모든 것을 sdbuild/output 디렉터리에 최종적인 부팅 가능 .img 파일로 조립합니다.

이 최종 이미지는 BalenaEtcher나 dd와 같은 도구를 사용하여 SD 카드에 기록할 수 있습니다.[1]

PYNQ sdbuild 흐름은 처음부터 OS를 빌드하는 시스템이 아니라, 정교한 오버레이 및 통합 시스템입니다. 이는 사전 빌드된 범용의 풍부한 기능의 Ubuntu 루트 파일 시스템과, PetaLinux를 통해 생성된 보드별 저수준 구성 요소(부트로더, 커널, 드라이버)를 영리하게 결합합니다. 모든 보드에 대해 전체 Ubuntu 배포판을 소스부터 재빌드하는 것은 매우 시간이 많이 걸리는 작업입니다. PYNQ 팀은 Ubuntu, Python, Jupyter 등을 포함하는 "보드에 구애받지 않는" 루트 파일 시스템을 미리 빌드해 둡니다.[81] PetaLinux 워크플로우는 하드웨어에 반드시

특화되어야 하는 부분, 즉 FSBL, U-Boot, 커널, 장치 트리만을 생성하는 데 사용됩니다.

sdbuild Makefile의 역할은 이 두 부분을 병합하는 것입니다. 범용 루트 파일 시스템을 가져와 사용자의 커스텀 BSP에서 나온 특정 부팅 구성 요소와 결합합니다. 이는 매우 효율적인 모델로, 커스텀 빌드 과정의 디버깅을 용이하게 합니다. 만약 보드가 부팅되지 않는다면 문제는 PetaLinux BSP에 있을 가능성이 높고, 부팅은 되지만 Jupyter가 작동하지 않는다면 문제는 루트 파일 시스템이나 sdbuild 통합 과정에 있을 수 있습니다.

Section 9: 결론**:** 사용자에서 아키텍트로 가는 길

9.1 워크플로우 종합

본 보고서는 PYNQ 소프트웨어 스택의 다층적 구조를 하드웨어부터 사용자 인터페이스까지 상세히 분석하고, 이를 바탕으로 커스텀 Zynq 보드에 PYNQ 환경을 구축하는 실질적인 3단계 절차를 제시했습니다. 이 과정은 Vivado를 통한 하드웨어 정의(XSA 출력), PetaLinux를 이용한 하드웨어 맞춤형 임베디드 Linux 생성(BSP 출력), 그리고 PYNQ의 sdbuild를 통한 최종 SD 카드 이미지 조립(IMG 출력)으로 요약됩니다. 각 단계는 명확한 입력과 출력을 가지며, 특정 도구의 역할을 중심으로 구성됩니다. 이 체계적인 접근법을 통해, 개발자는 단순한 하드웨어 보드를 Python으로 제어되는 강력하고 유연한 컴퓨팅 플랫폼으로 변환할 수 있습니다.

9.2 심화 주제 및 향후 방향

본 가이드에서 다룬 내용을 성공적으로 마스터한 개발자는 PYNQ 프레임워크의 단순한 사용자를 넘어, 자신만의 하드웨어 가속 시스템을 설계하는 아키텍트가 될 수 있는 기반을 다지게 됩니다. 더 나아가 다음과 같은 심화 주제들을 탐구할 수 있습니다.

- 프로덕션 환경을 위한 순수 **Yocto** 사용**:** PetaLinux는 개발 편의성에 중점을 두지만, 더 세밀한 제어와 보안 강화가 필요한 상용 제품 환경에서는 순수 Yocto 프로젝트를 직접 사용하여 시스템을 구축하는 것이 더 적합할 수 있습니다.[63]
- 복잡한 커스텀 오버레이 패키징**:** 자신만의 Python API와 Jupyter 노트북 예제를 포함하는 복잡한 커스텀 오버레이를 개발하고, 이를 다른 사용자들이 쉽게 설치하고 사용할 수 있도록 패키징하는 방법을 학습할 수 있습니다.[47]
- 고수준 합성**(HLS) IP** 통합**:** C/C++로 작성된 알고리즘을 Vitis HLS와 같은 도구를 사용하여

하드웨어 IP로 변환하고, 이를 PYNQ 워크플로우에 통합하여 소프트웨어 코드의 특정 부분을 하드웨어로 가속하는 방법을 탐구할 수 있습니다.[50]

이러한 심화 과정은 PYNQ가 제공하는 생산성의 진정한 잠재력을 발휘하게 하며, 개발자가 소프트웨어의 유연성과 하드웨어의 성능을 결합한 혁신적인 임베디드 시스템을 구축할 수 있도록 이끌 것입니다.

참고 자료

1. PYNQ-Z1 Reference Manual - Digilent, 9월 18, 2025에 액세스, https://digilent.com/reference/programmable-logic/pynq-z1/reference-manual
2. Python productivity for Zynq (Pynq) Documentation - Read the Docs, 9월 18, 2025에 액세스, https://buildmedia.readthedocs.org/media/pdf/pynq/v2.4/pynq.pdf
3. Overview of PYNQ project offering FPGA capabilities to Python and data engineers., 9월 18, 2025에 액세스, https://rkblog.dev/posts/python/overview-pynq-project-offering-fpga-capabilities-python-and-data-engineers/
4. PYNQ Overlays — Python productivity for Zynq (Pynq) v1.0 - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/v2.4/pynq_overlays.html
5. Blog #2: Third Eye for Blind - Deep Dive into ZYNQ & PYNQ - element14 Community, 9월 18, 2025에 액세스, https://community.element14.com/challenges-projects/design-challenges/eye-on-intelligence-challenge/b/blog/posts/blog-2-third-eye-for-blind---deep-dive-into-zynq-pynq
6. A PYNQ Evaluation Platform for FPGA Architectures of the Line Hough Transform - University of Strathclyde, 9월 18, 2025에 액세스, https://pureportal.strath.ac.uk/files/110341367/Northcote_etal_MWSCAS_2020_A_PYNQ_evaluation_platform_for_FPGA_architectures.pdf
7. PS/PL Interfaces — Python productivity for Zynq (Pynq) v1.0 - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/v2.1/overlay_design_methodology/pspl_interface.html
8. PS/PL Interfaces — Python productivity for Zynq (Pynq) v1.0 - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/v2.3/overlay_design_methodology/pspl_interface.html
9. PYNQ-Z1 - Digilent Reference, 9월 18, 2025에 액세스, https://digilent.com/reference/programmable-logic/pynq-z1/start
10. AUP PYNQ-Z2 - AMD, 9월 18, 2025에 액세스, https://www.amd.com/en/corporate/university-program/aup-boards/pynq-z2.html
11. PYNQ-Z2 Reference Manual v1.0 - Mouser Electronics, 9월 18, 2025에 액세스, https://www.mouser.com/datasheet/2/744/pynqz2_user_manual_v1_0-1525725.pdf

12. www.amd.com, 9월 18, 2025에 액세스, https://www.amd.com/en/developer/resources/kria-apps/vision-ai-dpu-pynq.html#:~:text=PYNQ%20is%20an%20open%2Dsource,more%20capable%20and%20intelligent%20systems.
13. PYNQ | Python Productivity for AMD Adaptive Computing platforms, 9월 18, 2025에 액세스, http://www.pynq.io/
14. PYNQ Framework on TityraCore Zynq® 7000 | Numato Lab Help Center, 9월 18, 2025에 액세스, https://numato.com/blog/pynq-framework-on-tityracore-zynq-7000/
15. Productivity of Embedded Designs with PYNQ - Avnet, 9월 18, 2025에 액세스, https://www.avnet.com/apac/products/cp/xilinx-pynq/reference-materials/productivity-of-embedded-designs-with-pynq/
16. Build and Program FPGA-Based Designs Quickly with Python and Jupyter Notebooks - DigiKey Polska, 9월 18, 2025에 액세스, https://www.digikey.pl/pl/articles/build-and-program-fpga-based-designs-quickly-python-jupyter-notebooks
17. Linux Boot Image Configuration — Embedded Design Tutorials 2021.1 documentation - GitHub Pages, 9월 18, 2025에 액세스, https://xilinx.github.io/Embedded-Design-Tutorials/docs/2021.1/build/html/docs/Introduction/Zynq7000-EDT/7-linux-booting-debug.html
18. ZynqLinux - HERO Documentation, 9월 18, 2025에 액세스, https://pulp-platform.org/hero/doc/software/host/zynqlinux/
19. Embedded Linux Primer - Mitchell Fang's Coding, 9월 18, 2025에 액세스, https://mfcoding.wordpress.com/linux/embedded-linux-primer/
20. Zynq booting - HDROB - SuS, 9월 18, 2025에 액세스, https://sus.ziti.uni-heidelberg.de/Projekte/HDROB/zynq_booting/
21. Boot and Configuration — Embedded Design Tutorials 2022.1 documentation - GitHub Pages, 9월 18, 2025에 액세스, https://xilinx.github.io/Embedded-Design-Tutorials/docs/2022.1/build/html/docs/Introduction/ZynqMPSoC-EDT/8-boot-and-configuration.html
22. Using Distro Boot - Xilinx Wiki - Confluence, 9월 18, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/749142017/Using+Distro+Boot+With+Xilinx+U-Boot
23. How Does Linux Know Where rootfs Is? | Baeldung on Linux, 9월 18, 2025에 액세스, https://www.baeldung.com/linux/rootfs-location
24. Hello PYNQ — The PYNQ Edition of the MicroZed Chronicles! | by Adam Taylor | Medium, 9월 18, 2025에 액세스, https://medium.com/@aptaylorceng/hello-pynq-the-pynq-edition-of-the-microzed-chronicles-55353a56c678
25. Python Productivity for Zynq, 9월 18, 2025에 액세스, https://www.xilinx.com/publications/events/xtd/05_Xilinx_Pynq_framework_to_support_Python_and_IIOT.pdf
26. Embedded Linux Boot Process in Resource Constrained Systems - Epteck GmbH, 9월 18, 2025에 액세스, https://epteck.com/embedded-linux-boot-process/
27. The Major Components of an Embedded Linux System - The New Stack, 9월 18,

2025에 액세스, https://thenewstack.io/the-major-components-of-an-embedded-linux-system/
28. Embedded Linux Design: File System and Bootloader - EEWeb, 9월 18, 2025에 액세스, https://www.eeweb.com/embedded-linux-design-file-system-and-bootloader/
29. Configuring and building RootFS for Zynq-7000 AP SoC - Xilinx Wiki, 9월 18, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842369/Build+Linux+for+Zynq-7000+AP+SoC+using+Buildroot?view=blog
30. Embedded Linux guys, report in. - EmbeddedRelated.com, 9월 18, 2025에 액세스, https://www.embeddedrelated.com/thread/16302/embedded-linux-guys-report-in
31. PYNQ-Z2 not use xdevcfg driver - Support, 9월 18, 2025에 액세스, https://discuss.pynq.io/t/pynq-z2-not-use-xdevcfg-driver/3152
32. Linux Drivers - Xilinx Wiki - Confluence, 9월 18, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/a/pages/18841873/Linux%2BDrivers
33. Solution Zynq PL Programming - Xilinx Wiki - Confluence, 9월 18, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841750/Solution+
34. Hello PYNQ — The PYNQ Edition of the MicroZed Chronicles! - Hackster.io, 9월 18, 2025에 액세스, https://www.hackster.io/news/hello-pynq-the-pynq-edition-of-the-microzed-chronicles-55353a56c678
35. Interrupt — Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/latest/pynq_libraries/interrupt.html
36. Uio device - Support - PYNQ, 9월 18, 2025에 액세스, https://discuss.pynq.io/t/uio-device/1801
37. PYNQ - AIoT Lab, 9월 18, 2025에 액세스, https://www.aiotlab.org/teaching/fpga/1_PYNQ_Workshop.pdf
38. pynq - PyPI, 9월 18, 2025에 액세스, https://pypi.org/project/pynq/
39. PYNQ Libraries — Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/latest/pynq_libraries.html
40. PYNQ MicroBlaze Subsystem - Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/latest/overlay_design_methodology/pynq_microblaze_subsystem.html
41. Overlay Design — Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/latest/overlay_design_methodology/overlay_design.html
42. Loading an Overlay — Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/latest/pynq_overlays/loading_an_overlay.html
43. Blog #5: CNN HW Accelerator for Handwriting Recognition - Integrating the HW Accelerator as a PYNQ Overlay - element14 Community, 9월 18, 2025에 액세스, https://community.element14.com/challenges-projects/design-challenges/eye-on

-intelligence-challenge/b/blog/posts/blog-5-integrating-the-hw-accelerator-as-a-pynq-overlay
44. PYNQ, Partial Reconfiguration, Part 2 - DJ Park, 9월 18, 2025에 액세스, https://dj-park.github.io/posts/2022/1/PYNQ_PR_2/
45. Overlay Design — Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/v2.7.0/overlay_design_methodology/overlay_design.html
46. Overlay — Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/latest/pynq_libraries/overlay.html
47. Python Overlay API - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/v2.4/overlay_design_methodology/python_overlay_api.html
48. Overlay Tutorial — Python productivity for Zynq (Pynq) v1.0 - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/v2.0/overlay_design_methodology/overlay_tutorial.html
49. Overlay Tutorial — Python productivity for Zynq (Pynq) v1.0 - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/v2.4/overlay_design_methodology/overlay_tutorial.html
50. Overlay Tutorial — Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/latest/overlay_design_methodology/overlay_tutorial.html
51. Dynamically Creating Usable Python Objects - The Overlay Class - Learn - PYNQ, 9월 18, 2025에 액세스, https://discuss.pynq.io/t/dynamically-creating-usable-python-objects-the-overlay-class/762
52. PYNQ-Z2 셋업 - 청색현인의 방랑기 - 티스토리, 9월 18, 2025에 액세스, https://ithryn.tistory.com/entry/PYNQ-Z2-%EC%85%8B%EC%97%85
53. PYNQ Productivity With Python | PDF | Field Programmable Gate Array | Hardware Description Language - Scribd, 9월 18, 2025에 액세스, https://www.scribd.com/document/638236391/PYNQ-Productivity-With-Python
54. PYNQ-ZU Base overlay, 9월 18, 2025에 액세스, https://xilinx.github.io/PYNQ-ZU/overlays.html
55. Xilinx/PYNQ_Workshop - GitHub, 9월 18, 2025에 액세스, https://github.com/Xilinx/PYNQ_Workshop
56. Tutorial: Creating a hardware design for PYNQ - Learn, 9월 18, 2025에 액세스, https://discuss.pynq.io/t/tutorial-creating-a-hardware-design-for-pynq/145
57. Pynq 2.6 custom board image build method that works - Learn, 9월 18, 2025에 액세스, https://discuss.pynq.io/t/pynq-2-6-custom-board-image-build-method-that-works/2894
58. PYNQ Edition! Creating Custom Overlays - Hackster.io, 9월 18, 2025에 액세스,

https://www.hackster.io/news/pynq-edition-creating-custom-overlays-8543f45eccb1
59. Step 1: Create the Vivado Hardware Design and Generate XSA — Vitis™ Tutorials 2021.1 documentation - GitHub Pages, 9월 18, 2025에 액세스, https://xilinx.github.io/Vitis-Tutorials/2021-1/build/html/docs/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104/step1.html
60. Xilinx Wiki - Confluence, 9월 18, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/524189697
61. PetaLinux Vivado XSA compilation issue : r/FPGA - Reddit, 9월 18, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/1mzxkya/petalinux_vivado_xsa_compilation_issue/
62. 2.2. Export Hardware to SDK — 802.11 MAC/PHY User Guide documentation, 9월 18, 2025에 액세스, https://support.mangocomm.com/docs/wlan-user-guide-v2/usage/sdk_export.html
63. Yocto or PetaLinux? - Cambridge Network, 9월 18, 2025에 액세스, https://www.cambridgenetwork.co.uk/news/yocto-or-petalinux
64. Yocto or PetaLinux? | Article - Plextek, 9월 18, 2025에 액세스, https://www.plextek.com/articles/yocto-or-petalinux/
65. Project Yocto vs Petalinux - Adaptive Support - AMD, 9월 18, 2025에 액세스, https://adaptivesupport.amd.com/s/question/0D52E00006iHwoTSAS/project-yocto-vs-petalinux?language=en_US
66. PetaLinux Tutorial+Demo - CERN Indico, 9월 18, 2025에 액세스, https://indico.cern.ch/event/799275/contributions/3413736/attachments/1861349/3059191/Petalinux_TutorialDemo.pdf
67. Creating a Custom Petalinux Image for Pynq-Z2 : r/FPGA - Reddit, 9월 18, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/1ao89df/creating_a_custom_petalinux_image_for_pynqz2/
68. Adam Taylor's MicroZed Chronicles, Part 219, UltraZed Edition: How to Build PetaLinux for the Zynq UltraScale+ MPSoC - Adaptive Support, 9월 18, 2025에 액세스, https://adaptivesupport.amd.com/s/article/799417
69. Step 2: Create the Software Components with PetaLinux — Vitis™ Tutorials 2022.1 documentation - GitHub Pages, 9월 18, 2025에 액세스, https://xilinx.github.io/Vitis-Tutorials/master/docs-jp/docs/Vitis_Platform_Creation/Design_Tutorials/03_Edge_VCK190/step2.html
70. Path to Programmable 3 - Training Blog 3 - Petalinux Overview, customization for custom hardware design and "hello ultra96v2" from python - element14 Community, 9월 18, 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
71. Xilinx/meta-jupyter - GitHub, 9월 18, 2025에 액세스, https://github.com/Xilinx/meta-jupyter

72. Deploying PYNQ and Jupyter with Petalinux - Learn, 9월 18, 2025에 액세스, https://discuss.pynq.io/t/deploying-pynq-and-jupyter-with-petalinux/677
73. Xilinx Wiki - Confluence, 9월 18, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/143589440/Executing+C+code+from+Python+on+Linux+%28Petalinux%29+on+ZCU102
74. petalinux python adding modules - Adaptive Support - AMD, 9월 18, 2025에 액세스, https://adaptivesupport.amd.com/s/question/0D52E00006hpmlvSAA/petalinux-python-adding-modules?language=en_US
75. How to add Python Libraries to Petalinux - Adaptive Support - AMD, 9월 18, 2025에 액세스, https://adaptivesupport.amd.com/s/question/0D52E000073GgHeSAK/how-to-add-python-libraries-to-petalinux?language=en_US
76. Getting Started with PetaLinux - Digilent Reference, 9월 18, 2025에 액세스, https://digilent.com/reference/software/petalinux
77. petalinux-build Examples - 2021.1 English - UG1144, 9월 18, 2025에 액세스, https://docs.amd.com/r/2021.1-English/ug1144-petalinux-tools-reference-guide/petalinux-build-Examples
78. Building Petalinux - GitHub Pages, 9월 18, 2025에 액세스, https://xilinx.github.io/vmk180-trd/2021.1/build/html/docs/build_petalinux.html
79. PYNQ Edition! Building PYNQ Images - Hackster.io, 9월 18, 2025에 액세스, https://www.hackster.io/news/pynq-edition-building-pynq-images-4f0ec30e172f
80. Enclustra board: how to create .bsp file to build PYNQ image - Support, 9월 18, 2025에 액세스, https://discuss.pynq.io/t/enclustra-board-how-to-create-bsp-file-to-build-pynq-image/447
81. PYNQ SD Card image — Python productivity for Zynq (Pynq) - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/latest/pynq_sd_card.html
82. PYNQ SD Card — Python productivity for Zynq (Pynq) v1.0 - Read the Docs, 9월 18, 2025에 액세스, https://pynq.readthedocs.io/en/v2.3/pynq_sd_card.html
83. Define Custom Pynq-Z2 Board with SoC Blockset - MATLAB & Simulink - MathWorks, 9월 18, 2025에 액세스, https://www.mathworks.com/help/soc/ug/define-custom-pynq-z2-board-with-socb.html
84. PYNQ Edition! Creating Custom Overlays | by Adam Taylor | Medium, 9월 18, 2025에 액세스, https://medium.com/@aptaylorceng/pynq-edition-creating-custom-overlays-8543f45eccb1
85. PYNQ: Python Productivity for Zynq - Dustin Richmond, 9월 18, 2025에 액세스, https://www.dustinrichmond.com/pynq/
86. Blog #3: Third Eye for Blind - How to Make and Modify a PYNQ Overlay, 9월 18, 2025에 액세스, https://community.element14.com/challenges-projects/design-challenges/eye-on-intelligence-challenge/b/blog/posts/blog-3-third-eye-for-blind---how-to-make-and-modify-a-pynq-overlay