적응형 무선 아키텍처: Zynq-7000 SoC 기반 SDR 플랫폼에서의 Dynamic Function eXchange 종합 가이드

I. 현대 소프트웨어 정의 라디오(SDR)에서 DFX의 전략적 필요성

현대 무선 통신 시스템의 복잡성과 요구사항이 기하급수적으로 증가함에 따라, 단순히 소프트웨어로 기능을 정의하는 것을 넘어 하드웨어 자체의 적응성을 확보하는 것이 중요한 과제로 부상했습니다. AMD의 Dynamic Function eXchange (DFX) 기술은 이러한 요구에 대한 핵심적인 해답을 제공합니다. DFX는 소프트웨어 정의 라디오(SDR)를 단순한 '소프트웨어 정의' 장치에서 실시간으로 하드웨어 구조를 변경할 수 있는 '하드웨어 적응형' 시스템으로 변모시키는 강력한 패러다임입니다. 본 섹션에서는 최신 무선 통신 환경의 역동적인 요구사항 속에서 DFX가 제공하는 전략적 이점을 심층적으로 분석합니다.

1.1. 개념적 프레임워크: AMD SoC의 "칩-속-칩(Chip-within-a-Chip)" 패러다임

Dynamic Function eXchange는 부분 재구성(Partial Reconfiguration, PR) 기술의 진화된 형태로, 시스템의 나머지 부분이 중단 없이 동작하는 동안 FPGA 패브릭의 특정 영역을 동적으로 수정할 수 있게 해줍니다.[1] 이 기술은 안정적으로 검증된 정적 플랫폼이 동적 애플리케이션 공간을 호스팅하는 "칩-속-칩" 모델을 구현합니다.[5] 이 모델은 세 가지 핵심 요소로 구성됩니다.

1. 정적 영역**(Static Region):** 프로세서 시스템 인터페이스, 메모리 컨트롤러, 핵심 I/O와 같이 시스템 운영에 필수적인 기반 구조를 포함하며, 재구성 과정 중에도 항상 활성 상태를 유지합니다.
2. 재구성 파티션**(Reconfigurable Partition, RP):** 프로그래머블 로직(PL) 내에 물리적으로 예약된 영역으로, 새로운 하드웨어 모듈이 로드될 '슬롯' 또는 '소켓' 역할을 합니다.[6]
3. 재구성 모듈**(Reconfigurable Module, RM):** 특정 기능을 수행하는 실제 로직(HDL

넷리스트)으로, 재구성 파티션에 동적으로 로드됩니다. 하나의 RP에는 여러 개의 서로 다른 RM이 번갈아 가며 로드될 수 있습니다.[7]

이러한 구조는 기존의 SoC-FPGA 활용 방식을 근본적으로 변화시킵니다. 전통적인 방식에서 PL은 고정된 기능의 가속기로 사용되며, 처리 시스템(PS)은 정적인 하드웨어 파이프라인에 특정 작업을 오프로드합니다. 반면, DFX는 이러한 고정된 하드웨어-소프트웨어 경계를 허물고, PL을 재구성 가능한 리소스의 유동적인 풀(pool)로 전환시킵니다. 이제 PS는 하드웨어 기능을 마치 소프트웨어 라이브러리처럼 필요에 따라 로드하고 교체할 수 있습니다. 이는 단일 실리콘 칩이 운영 수명 동안 여러 개의 고유한 하드웨어 정체성을 가질 수 있게 하는 패러다임의 전환을 의미하며, 시스템의 유연성과 효율성을 극대화합니다.

1.2. 다중 표준 및 다중 모드 민첩성: PHY 계층의 실시간 교체

SDR의 본질은 유연성에 있으며, 이는 전통적으로 FPGA 또는 DSP 기반의 디지털 백엔드에서 구현되었습니다.[9] DFX는 이러한 유연성을 하드웨어 수준에서 극대화하여, 전례 없는 적응성을 제공합니다. 예를 들어, 하나의 SDR 장치가 고속 데이터 전송을 위한 최신 5G NR 파형과 위성 통신을 위한 견고한 레거시 BPSK/QPSK 파형을 모두 지원해야 하는 시나리오를 생각해 볼 수 있습니다. 기존 방식으로는 두 파형 처리 로직을 모두 FPGA 패브릭에 구현해야 하므로 상당한 리소스를 소모하게 됩니다. 하지만 DFX를 사용하면, 활성화된 통신 링크의 요구에 따라 PS가 적절한 RM(예: 5G NR 채널 필터 및 변조기 RM 또는 BPSK/QPSK 변조기 RM)을 동적으로 로드할 수 있습니다.[7]

이러한 전환에서 가장 중요한 요소는 재구성 시간입니다. FPGA 전체를 재구성하는 데는 수 초가 소요될 수 있지만, DFX를 통한 부분 재구성은 수십에서 수백 밀리초 내에 완료됩니다.[7] 이처럼 빠른 전환 속도는 거의 실시간에 가까운 모드 변경을 가능하게 하여, 하나의 SDR 장비가 끊김 없이 다양한 통신 표준과 모드를 오가며 동작할 수 있도록 합니다. 이는 동적으로 변화하는 스펙트럼 환경에 대응하거나, 여러 통신 프로토콜을 동시에 지원해야 하는 군사 및 상업용 애플리케이션에서 결정적인 이점을 제공합니다.

1.3. SWaP-C (크기, 무게, 전력 및 비용) 최적화

DFX는 "시간 다중화 실리콘 사용(time-multiplexed silicon usage)"이라는 개념을 통해 시스템 리소스 효율성을 최적화합니다.[5] 현재 작업에 필요한 하드웨어 기능만 FPGA에 로드함으로써, 사용되지 않는 로직은 물리적으로 존재하지 않게 되어 동적 및 정적 전력 소비를 원천적으로 차단합니다.[1] 이는 시스템의 전체 전력 소모량을 직접적으로 감소시키는 효과를 가져옵니다.

이러한 시간 다중화 접근 방식은 설계자가 더 작고 저렴한 FPGA를 사용하여 시스템 요구사항을

충족할 수 있게 해줍니다. 가능한 모든 하드웨어 기능을 동시에 수용할 수 있는 대규모 FPGA를 선택하는 대신, 더 작은 디바이스를 선택하고 필요에 따라 기능을 교체하는 방식으로 시스템을 설계할 수 있습니다.[3] 이는 최종 제품의 비용(Cost)과 물리적 크기(Size), 무게(Weight)를 줄이는 데 직접적으로 기여하며, 특히 SWaP-C 제약이 엄격한 항공우주, 국방, 휴대용 장비 분야에서 매우 중요합니다.

더 나아가, DFX는 전력 관리를 단순한 온/오프 제어에서 고차원적인 아키텍처 설계 도구로 격상시킵니다. 예를 들어, SDR은 광대역 스펙트럼 스캐닝을 위해 간단하고 저전력으로 동작하는 RM을 사용하는 '저전력 감시 모드'로 운영될 수 있습니다. 특정 관심 신호가 감지되면, PS는 재구성을 트리거하여 복잡한 복조기 및 디코더를 포함하는 '고성능 분석 모드' RM을 로드할 수 있습니다. 이처럼 시스템의 전력 프로파일은 더 이상 단일 최악의 시나리오 값에 고정되지 않고, 임무 단계에 따라 동적으로 변화하는 프로파일을 갖게 됩니다. 이는 배터리로 구동되거나 열 제약이 있는 플랫폼에서 시스템의 운영 시간을 연장하고 신뢰성을 높이는 핵심적인 3차 이점을 제공합니다.

1.4. 현장에서의 진화**:** 원활한 하드웨어 업데이트 및 알고리즘 개선

DFX는 기본 플랫폼과 가속 기능의 개발 및 배포 수명 주기를 분리할 수 있게 해줍니다.[6] 이는 이미 현장에 배치된 SDR 장비가 전체 시스템 리콜이나 위험 부담이 큰 전체 펌웨어 업데이트 없이도 부분 비트스트림을 통해 하드웨어 업데이트를 수신할 수 있음을 의미합니다.

이 기능은 진화하는 통신 표준에 대응하거나, 개선된 신호 처리 알고리즘(예: 더 효율적인 FFT 또는 FIR 필터 구현)을 현장 장비에 배포할 때 특히 강력합니다. 업데이트 프로세스는 재구성 영역에 한정되어 원자적으로(atomically) 수행되므로, 운영 중인 정적 시스템의 안정성을 해치지 않으면서 다운타임을 최소화할 수 있습니다.[1] 이를 통해 제품의 수명 주기를 연장하고, 새로운 위협이나 기술 변화에 신속하게 대응하며, 지속적인 성능 개선을 제공하는 '살아있는' 시스템을 구축할 수 있습니다.

II. 아키텍처 심층 분석**: Zynq-7000 SoC**에서의 **DFX**

DFX의 개념을 ANTSDR E310에 탑재된 Zynq-7000 SoC에서 구체적으로 구현하는 방법을 이해하기 위해서는, 이를 가능하게 하는 하드웨어 및 소프트웨어 메커니즘에 대한 깊이 있는 분석이 필요합니다. 본 섹션에서는 Zynq-7000 아키텍처의 특성과 DFX 구현의 핵심 요소들을 상세히 설명합니다.

2.1. PS-PL 공생 관계: 재구성 엔진으로서의 처리 시스템

Zynq-7000 아키텍처는 이기종 컴퓨팅의 본질을 담고 있으며, 듀얼 코어 ARM Cortex-A9 처리 시스템(PS)과 프로그래머블 로직(PL)을 단일 다이에 통합한 구조입니다.[12] 이 구조에서 PS는 단순히 PL을 사용하는 주체가 아니라, PL을 제어하는 마스터 역할을 수행합니다. PS는 PL의 구성 로직에 대한 특권적인 접근 권한을 가지며, 이러한 긴밀한 통합은 DFX를 SoC 환경에서 구현하는 핵심 동력입니다. 독립형 FPGA가 외부 컨트롤러를 필요로 하는 것과 달리, Zynq SoC는 PS 자체가 재구성 엔진이 됩니다.[1]

PS와 PL 간의 통신은 고대역폭 AXI 상호연결(interconnect)을 통해 이루어집니다. 제어 신호는 주로 AXI-Lite 인터페이스를 통해 전달되며, 대용량 데이터는 AXI-HP(High Performance) 또는 AXI-GP(General Purpose) 포트를 통해 전송됩니다.[15] DFX 설계에서 이러한 AXI 인터페이스들은 정적 영역과 동적으로 변화하는 RM 간의 안정적인 통신 채널을 형성하는 기반이 됩니다.

2.2. 구성 경로: 파일 시스템에서 FPGA 패브릭까지, PCAP을 통해

Zynq SoC에서 PL을 구성하는 물리적인 메커니즘은 프로세서 구성 액세스 포트(Processor Configuration Access Port, PCAP)입니다. PCAP은 PS에서 접근 가능한 고대역폭 DMA(Direct Memory Access) 기반 포트로, PL 구성 데이터를 효율적으로 전송하는 역할을 합니다.[19]

DFX를 이용한 부분 재구성의 전체 워크플로우는 다음과 같습니다. 먼저, SD 카드와 같은 비휘발성 저장 매체에 저장된 부분 비트스트림 파일이 PS에 의해 DDR 메모리로 로드됩니다. 그 후, PS는 DMA 전송을 시작하여 PCAP이 DDR 메모리로부터 비트스트림을 읽어 PL의 구성 메모리로 스트리밍하도록 지시합니다. 이 모든 과정은 PS에서 실행되는 소프트웨어에 의해 완벽하게 제어되므로, 하드웨어 재구성은 본질적으로 소프트웨어 제어 작업이 됩니다.[3] 이는 운영 체제나 애플리케이션이 시스템의 하드웨어 기능을 실시간으로 변경할 수 있는 강력한 유연성을 제공합니다.

2.3. 소프트웨어 오케스트레이션: Linux FPGA Manager의 역할

PCAP을 베어메탈(bare-metal) 코드에서 직접 제어하는 것도 가능하지만, 표준적이고 가장 강력한 방법은 PetaLinux와 같은 고수준 운영 체제를 사용하는 것입니다. PetaLinux는 AMD가 맞춤화한 Yocto 프로젝트 기반의 임베디드 Linux 배포판으로, Zynq SoC 개발에 최적화되어 있습니다.[1]

Linux 커널은 FPGA 및 SoC를 프로그래밍하기 위한 **FPGA Manager**라는 일반 프레임워크를 제공합니다.[21] Zynq SoC를 위해,

zynq-fpga 드라이버가 이 프레임워크를 구현하며, sysfs 파일 시스템을 통해 사용자 공간(user-space) 애플리케이션에 제어 인터페이스를 노출합니다.[22] FPGA Manager를 사용하면 부분 비트스트림을 로드하는 작업이 sysfs 노드에 비트스트림 파일 이름을 쓰는 것만큼 간단해집니다. 커널 드라이버는 메모리 관리, PCAP 상호작용 등 모든 저수준 세부 사항을 추상화하여 처리합니다.[24]

이러한 소프트웨어 중심 접근 방식에서, Linux가 PL 내의 재구성 가능한 영역을 어떻게 인식하고 관리하는지에 대한 질문이 생깁니다. 그 해답은 **디바이스 트리(Device Tree)**에 있습니다. 기본 디바이스 트리는 시스템의 정적 하드웨어 구성을 설명합니다. DFX를 위해서는 **디바이스 트리 오버레이(Device Tree Overlays, DTOs)**라는 강력한 메커니즘이 사용됩니다. 각 RM은 자신을 설명하는 고유한 오버레이 파일(.dtbo)과 연관됩니다. FPGA Manager가 부분 비트스트림을 로드할 때, 해당 DTO를 함께 적용할 수 있습니다. 이 과정은 RM에 포함된 새로운 하드웨어에 필요한 드라이버를 동적으로 로드하고 커널에 등록합니다. 결과적으로, 비트스트림 로드는 단순히 하드웨어를 변경하는 것을 넘어, 운영 체제에 그 변화를 알리고 소프트웨어가 즉시 새로운 기능을 사용할 수 있도록 만드는 완벽한 통합 프로세스가 됩니다. 이 DTO 메커니즘[22]은 DFX를 임베디드 Linux 환경에서 관리 가능하고 확장 가능한 기능으로 만드는 핵심적인 연결 고리입니다.

2.4. 시스템 무결성 보장**:** 하드웨어 디커플링 및 인터페이스 관리

짧은 재구성 시간 동안 RP 내부의 로직은 정의되지 않은 상태에 놓이게 됩니다. 이 기간 동안 RP에서 나오는 모든 신호는 비정상적인 값을 가질 수 있으며, 이는 정적 시스템의 안정성을 심각하게 위협할 수 있습니다.

이러한 문제를 해결하기 위해, AMD는 **DFX Decoupler** 및 **DFX AXI Shutdown Manager**와 같은 특수 IP 코어를 제공합니다.[1] 이 IP들은 재구성 중에 RP를 정적 설계로부터 논리적으로 격리시켜, 글리치나 유효하지 않은 AXI 트랜잭션이 시스템 전체로 전파되는 것을 방지합니다. DFX Decoupler는 일반 신호 라인을 안전한 상태(예: 로직 0)로 고정하는 역할을 하며, DFX AXI Shutdown Manager는 AXI 버스 트랜잭션을 정상적으로 종료시키고 재구성 중에는 새로운 트랜잭션을 차단합니다.

또한, 견고한 DFX 설계를 위해서는 하드웨어 설계 관행도 매우 중요합니다. 정적 영역과 RP 경계를 넘나드는 모든 신호를 레지스터로 처리하는 것이 대표적인 예입니다.[2] 이는 경계에서의 타이밍 예측 가능성을 높이고, 재구성 중 발생하는 비동기적인 신호 변화로 인한 문제를 최소화합니다. 이러한 설계 원칙과 전용 IP의 활용은 동적으로 변화하는 시스템에서도 무결성과 안정성을 보장하는 데 필수적입니다.

III. 개념 증명(PoC) 튜토리얼: ANTSDR E310에서 DFX 구현하기

본 섹션은 ANTSDR E310 플랫폼에서 DFX 시스템을 구축하고 실행하기 위한 완전하고 독립적인 가이드를 제공합니다. 각 단계는 명확한 명령어와 상세한 설명을 포함하여, 사용자가 직접 따라하며 DFX의 동작 원리를 체득할 수 있도록 구성되었습니다.

1부: 개발 환경 및 프로젝트 준비

1.1. 툴체인 설치 및 구성

성공적인 DFX 프로젝트를 위해서는 특정 버전의 AMD 개발 도구가 필요합니다. ANTSDR 펌웨어 저장소는 특정 Vivado 및 PetaLinux 버전에 대한 의존성을 명시하고 있습니다.[25] 이 튜토리얼에서는 현대적이고 안정적인 버전(예: Vivado 및 PetaLinux 2022.1 이상)을 기준으로 진행하며, 사용자는 자신이 사용하는 펌웨어 저장소와 호환되는 버전을 설치해야 합니다. AMD 공식 웹사이트에서 Vivado Design Suite와 PetaLinux Tools를 다운로드하여 설치합니다. 설치 후에는 각 도구의 환경 설정 스크립트(

settings64.sh)를 소싱하여 터미널 환경에서 명령어를 사용할 수 있도록 준비해야 합니다.

1.2. ANTSDR 펌웨어 저장소 및 보드 파일

ANTSDR E310에 맞는 하드웨어 정의, 빌드 스크립트, 패치 등을 얻기 위해 공식 펌웨어 저장소를 복제해야 합니다. GitHub에서 MicroPhase/antsdr-fw-patch 또는 bkerler/antsdr_new와 같은 저장소를 git clone 명령어로 로컬 시스템에 복제합니다.[26] 이 저장소에는 Vivado 프로젝트 생성에 필요한 보드 파일이 포함되어 있을 수 있습니다. Vivado 보드 파일은 특정 개발 보드의 부품 번호, 인터페이스, 핀配置 등을 사전 정의하여 프로젝트 생성을 단순화하는 역할을 합니다.[29] 저장소의 안내에 따라 보드 파일을 Vivado 설치 경로 내의

data/boards/board_files 디렉토리에 복사합니다.

1.3. 플랫폼 개요

PoC 설계에 앞서, 대상 플랫폼인 ANTSDR E310의 핵심 사양을 이해하는 것이 중요합니다. 이는 설계 과정에서 리소스 예산과 제약 조건을 명확히 하는 데 도움이 됩니다.

표 **1: ANTSDR E310** 주요 플랫폼 사양

구성 요소	사양	출처
SoC	AMD Zynq-7000 XC7Z020-CLG400	[14]
처리 시스템 **(PS)**	듀얼 코어 ARM Cortex-A9 @ 766MHz	[14]
프로그래머블 로직 **(PL)**	85k 로직 셀	[14]
PS 메모리	1 GByte DDR3	[14]
RF 트랜시버	Analog Devices AD9361/AD9363	[14]
연결성	기가비트 이더넷, USB 2.0 OTG, USB-JTAG	[14]
부팅 미디어	QSPI 플래시 (32MB), Micro SD 카드	[14]

이 사양표는 단순한 정보 나열 이상의 의미를 가집니다. XC7Z020의 85k 로직 셀은 정적 영역과 가장 큰 RM의 복잡도를 결정하는 리소스 예산입니다. 1GB DDR3 메모리는 PetaLinux 운영 체제가 실행되고, 부분 비트스트림이 PCAP으로 전송되기 전에 임시 저장되는 공간입니다. 기가비트 이더넷과 SD 카드는 개발된 비트스트림을 장치로 전송하고 부팅하는 주요 수단입니다. 따라서 이 표는 PoC 설계의 제약 조건과 활용 가능한 자원을 규정하는 설계 지침서 역할을 합니다.

2부**: Vivado**를 이용한 하드웨어 플랫폼 설계

2.1. 기본 프로젝트 및 정적 시스템 생성

1. Vivado를 실행하고 ANTSDR E310 (부품 번호: xc7z020clg400-1)을 대상으로 하는 새 RTL 프로젝트를 생성합니다.
2. IP Integrator를 사용하여 새 블록 디자인을 생성합니다.
3. ZYNQ7 Processing System IP를 블록 디자인에 추가하고, Run Block Automation을 실행하여 DDR 및 고정 I/O에 대한 기본 구성을 적용합니다. 이 블록이 정적 디자인의 핵심이 됩니다.[15]
4. Zynq PS 블록을 더블 클릭하여 M_AXI_GP0 포트와 FCLK_CLK0를 활성화합니다.
5. PS-PL 간 데이터 루프백을 검증하기 위해 PL 영역에 AXI BRAM Controller와 Block Memory Generator IP를 추가합니다. Run Connection Automation을 실행하여 BRAM 컨트롤러를 PS의 M_AXI_GP0 포트에 연결합니다. 이 BRAM은 PS와 PL 간의 데이터 교환을 위한 공유 메모리 역할을 합니다.[18]

2.2. 재구성 파티션**(RP)** 정의

1. 블록 디자인에서 Tools -> Enable Dynamic Function eXchange를 선택하여 DFX 워크플로우를 활성화합니다.
2. DFX Wizard가 나타나면, Convert 탭을 선택하고 Instantiate를 클릭하여 재구성 파티션으로 사용할 계층 블록을 생성합니다. 이 블록의 이름을 rp_inst로 지정합니다.
3. rp_inst 블록의 인터페이스를 정의합니다. 입력용 AXI4-Stream 슬레이브 포트(S_AXIS), 출력용 AXI4-Stream 마스터 포트(M_AXIS), 그리고 클럭(ap_clk) 및 리셋(ap_rst_n) 포트를 추가합니다. 이 인터페이스는 모든 RM에서 동일하게 유지되어야 하는 '소켓' 규격입니다.
4. 생성된 rp_inst 블록의 S_AXIS와 M_AXIS 포트를 블록 디자인 외부로 노출시키기 위해 마우스 오른쪽 버튼을 클릭하고 Make External을 선택합니다. 이 튜토리얼에서는 외부 포트를 다시 내부적으로 연결하여 간단한 루프백을 구성합니다.
5. rp_inst의 클럭 및 리셋 포트를 PS에서 생성된 FCLK_CLK0 및 peripheral_aresetn에 연결합니다.

2.3. 재구성 모듈**(RM)** 생성

DFX의 핵심인 기능 교체를 시연하기 위해, 기능적으로 명확하게 구분되는 두 개의 간단한 RM을 생성합니다. 두 RM은 rp_inst에서 정의한 인터페이스와 정확히 일치하는 최상위 모듈 정의를 가져야 합니다.[7]

1. **RM1 ("Pass-through"):** 이 모듈은 입력받은 AXI-Stream 데이터를 아무런 처리 없이 그대로 출력으로 전달합니다. 데이터 무결성 및 루프백 경로의 기준선을 확인하는 역할을 합니다.
2. **RM2 ("Data Inverter"):** 이 모듈은 입력받은 AXI-Stream 데이터에 대해 비트 단위 NOT 연산을 수행한 후 출력으로 전달합니다. 이 모듈을 통해 하드웨어 기능이 성공적으로 교체되었음을 명확하게 확인할 수 있습니다.

DFX Wizard에서 Add Reconfigurable Module을 사용하여 각 RM에 대한 HDL 소스 파일을 추가하고, rm_passthrough와 rm_inverter라는 두 개의 RM을 생성합니다.

표 **2: PoC** 재구성 모듈 기능 개요

모듈 이름	기능	목적
rm_passthrough	output_data <= input_data	데이터 무결성 및 루프백 기능의 기준선 설정
rm_inverter	output_data <= NOT(input_data)	데이터 경로에서 검증 가능한 기능적 변화 제공

이 표는 PoC에서 교체될 두 가지 하드웨어 '정체성'의 동작을 명확히 정의합니다. 이는 검증 단계의 성공 기준을 설정하는 데 중요합니다. rm_passthrough가 로드되었을 때 PS가 읽어 들인 데이터는 쓴 데이터와 동일해야 하며, rm_inverter가 로드되었을 때는 비트 반전된 값이어야 합니다. 이러한 명확성은 튜토리얼의 이해를 돕는 데 필수적입니다.

2.4. 제약 조건 및 구현

1. DFX Wizard에서 Floorplan 탭으로 이동하여 rp_inst 파티션에 대한 Pblock을 생성합니다. Draw Pblock을 사용하여 디바이스 뷰에서 물리적 영역을 할당합니다. 이 영역은 모든 RM을 수용할 수 있을 만큼 충분히 커야 합니다.[2]
2. DFX 워크플로우는 HD.RECONFIGURABLE 속성을 RP 인스턴스에 자동으로 설정하여, 정적 영역과 RP 경계 간의 최적화를 방지하고 인터페이스를 고정시킵니다.[7]
3. Design Runs 창에서 DFX 구현 흐름을 실행합니다. Vivado는 먼저 정적 로직과 기본 RM(rm_passthrough)을 포함하는 config_1 (부모 실행)을 구현합니다. 그 후, 정적 로직의

배치 및 라우팅 정보를 재사용하여 두 번째 RM(rm_inverter)에 대한 config_2 (자식 실행)를 구현합니다.[37]

2.5. 비트스트림 생성 및 하드웨어 내보내기

1. 모든 구현이 성공적으로 완료되면, Tcl Console에서 write_bitstream -all 명령을 실행합니다. 이 명령은 config_1에 대한 전체 비트스트림과 각 RM(rm_passthrough, rm_inverter)에 대한 부분 비트스트림을 생성합니다.[6]
2. File -> Export -> Export Hardware 메뉴를 통해 정적 하드웨어 설계를 XSA(Xilinx Support Archive) 파일로 내보냅니다. 이 XSA 파일은 PetaLinux 프로젝트를 생성하는 데 사용될 하드웨어 플랫폼의 청사진입니다.[39]

3부**: PetaLinux**를 이용한 임베디드 **Linux** 시스템 빌드

3.1. PetaLinux 프로젝트 생성

1. 터미널에서 PetaLinux 환경을 설정합니다 (source <peta_linux_install_dir>/settings.sh).
2. petalinux-create --type project --template zynq --name antsdr_dfx 명령으로 새 PetaLinux 프로젝트를 생성합니다.
3. 생성된 프로젝트 디렉토리로 이동한 후, petalinux-config --get-hw-description=<path_to_xsa_file> 명령을 실행하여 Vivado에서 내보낸 하드웨어 플랫폼 정보를 프로젝트에 가져옵니다.[20]

3.2. 커널 및 루트 파일 시스템 구성

1. petalinux-config -c kernel 명령을 실행하여 커널 구성 메뉴로 들어갑니다. Device Drivers -> FPGA Configuration Framework에서 Xilinx FPGA Manager가 활성화되어 있는지 확인합니다. Zynq-7000의 경우, 이 옵션이 기본적으로 활성화되어 있을 수 있습니다.[21]
2. petalinux-config -c rootfs 명령으로 루트 파일 시스템 구성 메뉴로 들어갑니다. 이 튜토리얼에서는 기본 구성을 사용하지만, 필요한 경우 추가적인 사용자 공간 유틸리티를 포함시킬 수 있습니다.

3. 생성된 부분 비트스트림 파일들을 타겟 보드의 루트 파일 시스템에 포함시켜야 합니다. <proj_dir>/project-spec/meta-user/recipes-core/images/petalinux-image-append.bb 파일을 열고 다음 내용을 추가하여 비트스트림 파일들이 /lib/firmware 디렉토리에 복사되도록 합니다.

```
FILESEXTRAPATHS:prepend := "${THISDIR}/files:"

SRC_URI += " \
    file://rm_passthrough.bit.bin \
    file://rm_inverter.bit.bin \
"

do_install:append() {
    install -d ${D}/lib/firmware
    install -m 0644 ${WORKDIR}/rm_passthrough.bit.bin ${D}/lib/firmware/
    install -m 0644 ${WORKDIR}/rm_inverter.bit.bin ${D}/lib/firmware/
}
```

그리고 <proj_dir>/project-spec/meta-user/recipes-core/images/files/ 디렉토리를 생성한 후, Vivado에서 생성된 *.bit.bin 파일들을 이 디렉토리에 복사합니다.

3.3. SD 카드 이미지 빌드

1. petalinux-build 명령을 실행하여 커널, 루트 파일 시스템, 디바이스 트리 등 전체 시스템을 컴파일합니다.
2. 빌드가 완료되면, petalinux-package --boot --fsbl images/linux/zynq_fsbl.elf --fpga images/linux/system.bit --u-boot 명령을 사용하여 부팅에 필요한 BOOT.BIN 파일을 생성합니다.
3. 이제 images/linux/ 디렉토리에는 SD 카드로 부팅하는 데 필요한 모든 파일(BOOT.BIN, image.ub, boot.scr)이 준비되었습니다.[43]

4부**:** 배포**,** 실행 및 검증

4.1. 부팅 및 시스템 설정

1. FAT32로 포맷된 Micro SD 카드에 BOOT.BIN, image.ub, boot.scr 파일을 복사합니다.
2. ANTSDR E310의 부팅 모드 점퍼를 SD 카드 부팅으로 설정하고, SD 카드를 삽입합니다.
3. USB-UART 케이블을 연결하여 PC와 시리얼 콘솔 연결을 설정하고(Baud rate: 115200), 이더넷 케이블을 연결합니다.
4. 보드에 전원을 공급하면 PetaLinux가 부팅되는 과정을 시리얼 콘솔에서 확인할 수 있습니다.[34]

4.2. DFX 제어 애플리케이션

Linux가 부팅된 후, FPGA Manager의 sysfs 인터페이스를 사용하여 PL을 재구성할 수 있습니다. 다음 쉘 스크립트는 rm_inverter 모듈을 로드하는 예시입니다.

Bash

```
#!/bin/sh
# load_inverter.sh

BITSTREAM_FILE="rm_inverter.bit.bin"
FIRMWARE_PATH="/lib/firmware/$BITSTREAM_FILE"
FPGA_MGR_PATH="/sys/class/fpga_manager/fpga0"

if; then
    echo "Error: Bitstream file not found at $FIRMWARE_PATH"
    exit 1
fi

echo "Loading partial bitstream: $BITSTREAM_FILE"

# Set to partial reconfiguration mode
echo 1 > $FPGA_MGR_PATH/flags

# Write firmware name to trigger loading
echo $BITSTREAM_FILE > $FPGA_MGR_PATH/firmware

echo "Reconfiguration complete."
```

4.3. PS-PL 데이터 루프백 테스트

다음은 devmem 유틸리티를 사용하여 PS에서 PL의 BRAM에 데이터를 쓰고 읽어 루프백을 검증하는 간단한 쉘 스크립트 예시입니다. (BRAM 컨트롤러의 주소는 Vivado Address Editor에서 확인해야 하며, 여기서는 0x40000000으로 가정합니다.)

Bash

```
#!/bin/sh
# verify_loopback.sh

BRAM_ADDR=0x40000000
TEST_VAL_WRITE=0xAAAAAAAA

echo "Writing 0x$TEST_VAL_WRITE to BRAM at address $BRAM_ADDR"
devmem $BRAM_ADDR 32 $TEST_VAL_WRITE

echo "Reading from BRAM..."
TEST_VAL_READ=$(devmem $BRAM_ADDR 32)

echo "Value read: $TEST_VAL_READ"

# Note: This simple script assumes the PL logic automatically
# performs the loopback. A more robust C application would be
# needed to control the start/end of the transaction.
# For this PoC, we assume the loopback is continuous.

if; then
    echo "Verification PASSED: Pass-through module is active."
elif; then
    echo "Verification PASSED: Inverter module is active."
else
    echo "Verification FAILED: Unexpected value read."
fi
```

4.4. 검증 절차

1. ANTSDR E310을 부팅하고 시리얼 콘솔로 로그인합니다.
2. 초기 상태(기본 RM인 rm_passthrough가 로드됨)를 확인하기 위해 ./verify_loopback.sh를 실행합니다. Value read: 0xAAAAAAAA 및 "Pass-through module is active" 메시지가 출력되어야 합니다.
3. ./load_inverter.sh 스크립트를 실행하여 rm_inverter 모듈을 로드합니다.
4. 다시 ./verify_loopback.sh를 실행합니다. 이제 Value read: 0x55555555 및 "Inverter module is active" 메시지가 출력되어야 합니다.

이 결과는 하드웨어 기능이 Linux 사용자 공간의 간단한 명령을 통해 성공적으로 동적으로 교체되었음을 명확하게 증명합니다.

표 **3: DFX** 관리 및 검증을 위한 핵심 **Linux** 명령어

명령어	목적
dmesg | grep fpga	FPGA Manager 드라이버가 정상적으로 로드되었는지 확인
ls /sys/class/fpga_manager/fpga0/	DFX 제어를 위한 sysfs 인터페이스 탐색
echo 1 > /sys/class/fpga_manager/fpga0/flags	FPGA Manager를 부분 재구성 모드로 설정
echo rm_inverter.bit.bin > /sys/class/fpga_manager/fpga0/firmware	새로운 재구성 모듈의 로딩을 트리거
./verify_loopback.sh	현재 PL 기능성을 테스트하는 애플리케이션 실행

이 표는 Linux 사용자 공간과의 상호작용을 명확하게 보여주는 '치트 시트' 역할을 합니다. '소프트웨어 제어'라는 추상적인 개념을 구체적이고 실행 가능한 코드로 변환하여, DFX를 처음 접하는 개발자가 실험하고 학습하는 데 따르는 진입 장벽을 낮춰줍니다.

IV. 고급 구현 및 검증 기법

PoC를 통해 DFX의 기본 원리를 이해했다면, 실제 시스템 개발에서 마주할 수 있는 문제들을 해결하기 위한 고급 기법들을 학습할 필요가 있습니다. 본 섹션에서는 DFX 설계의 디버깅, 타이밍 마감, 컴파일 시간 단축과 관련된 실용적인 고급 방법론을 다룹니다.

4.1. Vivado Logic Analyzer (ILA)를 이용한 DFX 설계의 인시스템 디버깅

하드웨어 자체가 동적으로 변화하는 시스템을 디버깅하는 것은 상당한 도전 과제입니다. Vivado의 ILA(Integrated Logic Analyzer)를 활용하여 DFX 설계를 효과적으로 디버깅하는 두 가지 주요 전략이 있습니다.

- 전략 **1 (**정적 프로빙**):** ILA 코어를 정적 영역에 배치하고, RP로 들어가고 나오는 AXI-Stream 인터페이스에 프로브를 연결하는 방식입니다. 이 방법은 어떤 RM이 로드되더라도 RP 경계에서의 데이터 흐름을 일관되게 관찰할 수 있게 해줍니다. 이는 인터페이스 프로토콜 문제나 정적-동적 영역 간의 상호작용을 디버깅하는 데 가장 견고하고 신뢰성 있는 접근 방식입니다.[46]
- 전략 **2 (**동적 프로빙**):** 더 작은 ILA 코어를 각 RM 내부에 직접 인스턴스화하여 모듈 내부의 신호를 관찰하는 방식입니다. 이 접근법은 특정 RM의 내부 로직을 상세하게 디버깅하는 데 유용하지만, 디버깅할 모든 RM에 ILA 코어를 포함시켜야 하는 부담이 있습니다.[48]

합성 후 단계에서 Vivado의 "Set up Debug" 마법사를 사용하면, GUI를 통해 정적 디자인의 넷리스트에 ILA 코어를 쉽게 추가하고 RP 경계의 넷에 연결할 수 있습니다. 이 방법을 사용하면 RTL 코드를 수정하지 않고도 필요한 디버깅 로직을 삽입할 수 있어 효율적입니다.[47]

4.2. 재구성 경계에 대한 타이밍 마감 및 제약 조건 전략

정적 영역과 RP 사이의 인터페이스는 시스템 전체의 성능을 좌우하는 중요한 타이밍 경로입니다. Vivado의 DFX 흐름은 이 경계에 DONT_TOUCH와 같은 제약 조건을 자동으로 적용하여, 서로 다른 구성 간에 불일치를 유발할 수 있는 경계 교차 최적화를 방지합니다.[7]

설계자는 이 경로에 대한 명시적인 타이밍 제약 조건(예: set_max_delay)을 생성하여 Vivado가 타이밍을 정확하게 분석하고 마감할 수 있도록 해야 합니다. 또한, RP 경계를 물리적으로 가로지르는 라우팅 리소스인 '파티션 핀'의 위치와 수를 최적화하는 것도 중요합니다. 파티션 핀의 수가 너무 많거나 분산되어 있으면 라우팅 혼잡을 유발하고 타이밍 마감을 어렵게 만들 수 있습니다.[2]

DFX가 부과하는 이러한 제약 조건들은 역설적으로 더 나은 설계 관행을 유도합니다. 단일체 설계에서는 합성기가 계층 경계를 자유롭게 넘나들며 최적화를 수행할 수 있지만, DFX는 이를

금지합니다. 이는 설계자가 레지스터로 처리된 입력과 출력을 갖는 명확하고 잘 정의된 인터페이스를 사용하도록 강제합니다. 결과적으로, DFX는 스파게티처럼 얽힌 로직이 계층 경계를 넘나드는 것을 방지하여 전체 아키텍처를 더 견고하고 유지보수하기 쉽게 만듭니다. 이는 설계 품질을 향상시키는 중요한 2차적 이점입니다.

4.3. Abstract Shell 워크플로우 소개

복잡한 시스템에서 새로운 RM을 추가하거나 수정할 때마다 전체 정적 영역을 포함하여 재컴파일하는 것은 매우 시간 소모적인 작업입니다.[5] 이러한 문제를 해결하기 위해 AMD는

Abstract Shell이라는 고급 워크플로우를 제공합니다.

Abstract Shell 워크플로우는 정적 디자인을 한 번 성공적으로 구현한 후, Vivado가 해당 디자인에서 RP 인터페이스 정보만을 남기고 내부 로직의 대부분을 제거한 가벼운 '껍데기' 버전을 생성하는 방식입니다. 이 껍데기는 새로운 RM을 컴파일하는 데 필요한 최소한의 컨텍스트만 제공합니다. 따라서 개발자는 훨씬 작은 이 셸을 대상으로 새로운 RM을 컴파일함으로써, 컴파일 시간과 메모리 사용량을 극적으로 줄일 수 있습니다.[1]

또한, 이 워크플로우는 팀 기반 협업과 IP 보안을 강화하는 데에도 큰 이점을 가집니다. RM을 개발하는 팀이나 외부 협력업체에게 독점적인 IP가 포함된 전체 정적 디자인을 공유할 필요 없이, 기능이 제거된 Abstract Shell만 제공하면 되기 때문입니다. 이를 통해 민감한 설계 정보를 보호하면서 효율적인 병렬 개발이 가능해집니다.[1]

V. 결론 및 향후 전망

5.1. SDR 시스템의 전력 증강 요소로서의 DFX 요약

본 보고서는 AMD의 Dynamic Function eXchange 기술이 현대 SDR 시스템에 제공하는 혁신적인 가치를 다각도로 조명했습니다. DFX는 단순한 기능 추가를 넘어, SDR의 아키텍처 자체를 동적으로 변화시키는 핵심 기술입니다. 주요 이점을 요약하면 다음과 같습니다.

- 전례 없는 민첩성**:** 다중 표준 및 다중 모드 통신 환경에서 실시간에 가깝게 물리 계층(PHY) 하드웨어를 교체하여 단일 플랫폼으로 다양한 요구사항에 대응할 수 있습니다.
- **SWaP-C** 최적화**:** 실리콘 리소스를 시간적으로 다중화하여 사용함으로써, 시스템의 크기,

무게, 전력 소비 및 비용을 획기적으로 절감합니다.
- 현장 업그레이드 가능성: 배치된 장비의 하드웨어를 원격으로 안전하게 업데이트하여 제품 수명 주기를 연장하고, 진화하는 기술과 위협에 지속적으로 대응할 수 있습니다.

결론적으로, DFX는 더 이상 일부 고급 사용자를 위한 난해한 기능이 아니라, 차세대 지능형 적응 무선 시스템을 구현하기 위한 필수적인 기술로 자리매김하고 있습니다.

5.2. 더 복잡한 구현으로의 경로

본 보고서에서 제시된 PoC는 DFX의 기본적인 원리를 보여주기 위한 출발점입니다. 이 기초 위에, 다음과 같은 더 복잡하고 실용적인 SDR 애플리케이션을 구현할 수 있습니다.

- 동적 필터 체인: 신호 조건에 따라 다양한 디지털 필터 체인(예: FIR, IIR, 다상 필터)을 포함하는 RM을 동적으로 교체하여 수신 성능을 최적화할 수 있습니다.[6]
- 적응형 스펙트럼 분석: 필요한 해상도와 속도에 따라 각기 다른 포인트 수의 FFT 엔진 RM을 로드하여 스펙트럼 분석 기능을 동적으로 조절할 수 있습니다.[6]
- 임무 특화 가속: 특정 임무에 필요한 하드웨어 가속기(예: AES 암호화, 머신러닝 추론을 위한 DPU)를 필요할 때만 로드하여 리소스 효율성을 극대화할 수 있습니다.[6]
- 다중 슬롯 아키텍처: 여러 개의 독립적인 RP를 설계하여, 다양한 기능의 RM들을 실시간으로 조합함으로써 완전히 새로운 하드웨어 파이프라인을 동적으로 구성할 수 있습니다.[6]

이러한 고급 애플리케이션들은 DFX가 제공하는 유연성을 최대한 활용하여, 변화하는 환경과 요구사항에 지능적으로 적응하는 진정한 의미의 '인지 무선(Cognitive Radio)' 시스템을 향한 중요한 발판을 마련할 것입니다.

참고 자료

1. Dynamic Function eXchange (DFX) - AMD, 9월 27, 2025에 액세스, https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/dynamic-function-exchange.html
2. Introduction to Dynamic Function eXchange - 2025.1 English - UG909, 9월 27, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Introduction-to-Dynamic-Function-eXchange
3. Understanding Partial Reconfiguration with Vivado - 01signal.com, 9월 27, 2025에 액세스, https://www.01signal.com/vendor-specific/xilinx/partial-reconfiguration/part1-introduction/
4. Vivado Design Suite User Guide: Partial Reconfiguration (UG909) - ivPCL, 9월 27, 2025에 액세스,

http://ivpcl.unm.edu/ivpclpages/Research/drastic/PRWebPage/ug909-vivado-partial-reconfiguration.pdf
5. Solution Efficiencies for Dynamic Function eXchange Using Abstract Shells - AMD, 9월 27, 2025에 액세스, https://download.amd.com/docnav/documents/aem/white_papers-wp533-abstract-shell.pdf
6. Dynamic Function eXchange (DFX) — Kria SOM DFX Examples 1.0 documentation - GitHub Pages, 9월 27, 2025에 액세스, https://xilinx.github.io/kria-apps-docs/dfx.html
7. Dynamic Function eXchange - Technology Blogs, 9월 27, 2025에 액세스, https://blog.abbey1.org.uk/index.php/technology/dynamic-function-exchange
8. (PDF) Partial reconfiguration on FPGAs in practice — Tools and applications, 9월 27, 2025에 액세스, https://www.researchgate.net/publication/254039951_Partial_reconfiguration_on_FPGAs_in_practice_-_Tools_and_applications
9. Software Defined Radios (SDR) for Aerospace and Defence - everything RF, 9월 27, 2025에 액세스, https://www.everythingrf.com/community/sdrs-for-aerospace-and-defence
10. SDR's Impact Across Various Domains: Enhancing Wireless Communication Flexibility, 9월 27, 2025에 액세스, https://yttek.com/sdrs-impact-across-various-domains-enhancing-wireless-communication-flexibility/
11. One of the coolest FPGA Technologies: AMDs DFX - Hackster.io, 9월 27, 2025에 액세스, https://www.hackster.io/marco13/one-of-the-coolest-fpga-technologies-amds-dfx-654e04
12. Zynq UltraScale+ MPSoC Data Sheet: Overview (DS891) - AMD, 9월 27, 2025에 액세스, https://www.amd.com/content/dam/xilinx/support/documents/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf
13. AMD Zynq™ UltraScale+™ MPSoCs, 9월 27, 2025에 액세스, https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-mpsoc.html
14. MicroPhase ANTSDR E310 AD9361 Software Defined Radio Open Source SDR ZYNQ7020, 9월 27, 2025에 액세스, https://www.ebay.com/itm/296798053980
15. Start to finish: a PL+PS design guide for Zynq UltraScale+ and PetaLinux (with UIO and interrupts from RTL or custom IP) : r/FPGA - Reddit, 9월 27, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/1mgmbyh/start_to_finish_a_plps_design_guide_for_zynq/
16. Exploring the PS-PL AXI interfaces on Zynq UltraScale+ MPSoC - j-marjanovic.io, 9월 27, 2025에 액세스, https://j-marjanovic.io/exploring-the-ps-pl-axi-interfaces-on-zynq-ultrascale-mpsoc.html
17. Zynq UltraScale+ MPSoC - System Performance Modelling - Xilinx Wiki - Confluence, 9월 27, 2025에 액세스,

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842421/Zynq+UltraScale+MPSoC+-+System+Performance+Modelling
18. Simple Zynq PS/PL communication : r/FPGA - Reddit, 9월 27, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/5wejo5/simple_zynq_pspl_communication/
19. PS Bring-up with PL Configuration Example - UG585 - AMD Technical Information Portal, 9월 27, 2025에 액세스, https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM/PS-Bring-up-with-PL-Configuration-Example
20. FPGA Manager Full PL Programming - 2025.1 English - UG1144, 9월 27, 2025에 액세스, https://docs.amd.com/r/en-US/ug1144-petalinux-tools-reference-guide/FPGA-Manager-Full-PL-Programming
21. Linux Drivers - Xilinx Wiki - Spaces - Confluence, 9월 27, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/a/pages/18841873/Linux%2BDrivers
22. Solution ZynqMP PL Programming - Xilinx Wiki - Confluence, 9월 27, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841847/Solution+ZynqMP+PL
23. Solution ZynqMP PL Programming - Xilinx Wiki - Confluence, 9월 27, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/display/A/Solution+ZynqMP+PL+Programming
24. PL Programming - Xilinx Wiki - Confluence, 9월 27, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/1188397412
25. MicroPhase/antsdr-fw: ANTSDR Firmware - GitHub, 9월 27, 2025에 액세스, https://github.com/MicroPhase/antsdr-fw
26. black-pigeon/antsdr-fw - GitHub, 9월 27, 2025에 액세스, https://github.com/black-pigeon/antsdr-fw
27. MicroPhase/antsdr-fw-patch: Repository of antsdr firmware make - GitHub, 9월 27, 2025에 액세스, https://github.com/MicroPhase/antsdr-fw-patch
28. bkerler/antsdr_new: Latest firmware for antsdr E310 based on PlutoSDR - GitHub, 9월 27, 2025에 액세스, https://github.com/bkerler/antsdr_new/
29. Xilinx/XilinxBoardStore - GitHub, 9월 27, 2025에 액세스, https://github.com/Xilinx/XilinxBoardStore
30. Digilent/vivado-boards - GitHub, 9월 27, 2025에 액세스, https://github.com/Digilent/vivado-boards
31. Install Digilent's Board Files - Digilent Reference, 9월 27, 2025에 액세스, https://digilent.com/reference/programmable-logic/guides/install-board-files
32. What is a Constraints File? - Digilent Reference, 9월 27, 2025에 액세스, https://digilent.com/reference/programmable-logic/guides/vivado-xdc-file
33. MicroPhase ANTSDR E310 AD9363 SDR Development Board for ADI Pluto Communications | eBay, 9월 27, 2025에 액세스, https://www.ebay.com/itm/355222519305
34. ANTSDR Unpacking and Test Rev. 1.1, 9월 27, 2025에 액세스, https://down.hgeek.com/download/93159/93159-E310-AD9361-Rev-1-1-English-M

anual.pdf
35. Vivado Design Suite User Guide Design Flows Overview, 9월 27, 2025에 액세스, https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug892-vivado-design-flows-overview.pdf
36. Zynq-7000 AP SoC - Performance - Ethernet Packet Inspection - Xilinx Wiki - Confluence, 9월 27, 2025에 액세스, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841889/Zynq-7000%2BAP%2BSoC%2B-%2BPerformance%2B-%2BEthernet%2BPacket%2BInspection%2B-%2BBare%2BMetal%2B-%2BRedirecting%2BPackets%2Bto%2BPL%2BTech%2BTip
37. Introduction - 2025.1 English - UG909, 9월 27, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Introduction
38. Vivado Design Suite User Guide: Dynamic Function eXchange (UG909) - 2025.1 English, 9월 27, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration
39. Dynamic Function Exchange with ZYNQ Ultracale+ : Part 5: Vivado Outputs and starting Vitis, 9월 27, 2025에 액세스, https://www.youtube.com/watch?v=O6gasw5D4cA
40. Design Flow for a Custom FPGA Board in Vivado and PetaLinux | by Whitney Knitter, 9월 27, 2025에 액세스, https://medium.com/@whitneyknitter/design-flow-for-a-custom-fpga-board-in-vivado-and-petalinux-b998c0b4f9f7
41. Using Avnet Build Scripts to Build a PetaLinux BSP (2020.1 and later), 9월 27, 2025에 액세스, https://community.element14.com/technologies/fpga-group/b/blog/posts/using-avnet-build-scripts-to-build-a-petalinux-bsp-2020-1-and-later
42. PetaLinux Tools Documentation Reference Guide, 9월 27, 2025에 액세스, https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_2/ug1144-petalinux-tools-reference-guide.pdf
43. Building and Debugging Linux Applications for Zynq-7000 SoCs - GitHub Pages, 9월 27, 2025에 액세스, https://xilinx.github.io/Embedded-Design-Tutorials/docs/2021.1/build/html/docs/Introduction/Zynq7000-EDT/4-linux-for-zynq.html
44. Building and Debugging Linux Applications for Zynq 7000 SoCs - 2025.1 English - UG1165, 9월 27, 2025에 액세스, https://docs.amd.com/r/en-US/ug1165-zynq-embedded-design-tutorial/Building-and-Debugging-Linux-Applications-for-Zynq-7000-SoCs
45. Building and Debugging Linux Applications for Zynq-7000 SoCs — Embedded Design Tutorials 2022.2 documentation - GitHub Pages, 9월 27, 2025에 액세스, https://xilinx.github.io/Embedded-Design-Tutorials/docs/2022.2/build/html/docs/Introduction/Zynq7000-EDT/4-linux-for-zynq.html
46. Debugging Dynamic Function eXchange (DFX) Designs in Vivado Hardware Manager - 2025.1 English - UG908, 9월 27, 2025에 액세스, https://docs.amd.com/r/en-US/ug908-vivado-programming-debugging/Debugging-Dynamic-Function-eXchange-DFX-Designs-in-Vivado-Hardware-Manager
47. Debugging in Vivado Tutorial - 2025.1 English - UG936, 9월 27, 2025에 액세스,

https://docs.amd.com/r/en-US/ug936-vivado-tutorial-programming-debugging/Debugging-in-Vivado-Tutorial
48. Vivado Design Suite Tutorial: Dynamic Function eXchange, 9월 27, 2025에 액세스, https://static.eetrend.com/files/2021-12/wen_zhang_/100556482-232279-ug947-vivado-partial-reconfiguration-tutorial.pdf
49. Vivado ILA(Integrated Logic Analyzer) : r/FPGA - Reddit, 9월 27, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/xhi4i4/vivado_ilaintegrated_logic_analyzer/