AMD Dynamic Function eXchange(DFX) 완벽 트러블슈팅 가이드

섹션 1: 견고한 DFX 설계를 위한 기본 원칙

이 서론 섹션에서는 이후 설계 흐름에서 특정 문제가 발생하는 근본적인 이유를 이해하는 데 필요한 핵심 개념을 다룹니다. DFX를 단순한 기능이 아닌, 단일 구조의 플랫 설계 방식에서 벗어나 물리적 설계 패러다임의 전환을 요구하는 기술로 접근합니다.

1.1 DFX 패러다임: 정적 로직과 동적 로직에 대한 심층 분석

DFX(Dynamic Function eXchange)는 부분 재구성(Partial Reconfiguration, PR) 기술을 계승한 개념으로, 시스템이 동작하는 동안 FPGA의 특정 영역을 동적으로 재프로그래밍할 수 있게 해주는 강력한 기술입니다.[1] 이 기술의 핵심은 설계를 정적(static) 영역과 하나 이상의 재구성 가능 파티션(Reconfigurable Partitions, RP)으로 분할하는 것입니다.[4] RP가 재구성되는 동안에도 정적 로직은 중단 없이 계속해서 동작합니다.

이러한 유연성은 시스템의 모드나 임무 단계에 따라 기능을 교체하거나, 여러 슬롯 기반 설계에서 고유한 기능 조합을 동적으로 구성하는 등 흥미로운 시스템 레벨의 사용 사례를 가능하게 합니다.[4] 예를 들어, Kria SOM 보드에서는 DFX를 활용하여 암호화(AES), 신호 처리(FFT), 필터(FIR)와 같은 다양한 하드웨어 가속기 모듈을 필요에 따라 동적으로 로드하고 교체할 수 있습니다.[4]

DFX 설계에서 발생하는 대부분의 문제와 오류의 근원은 본질적으로 다른 두 도메인, 즉 정적 영역과 동적 영역 간의 물리적 및 논리적 인터페이스를 관리하는 데서 비롯됩니다. 따라서 성공적인 DFX 구현을 위해서는 이 두 영역 간의 명확한 경계를 정의하고, 이 경계를 넘나드는 모든 신호와 클럭을 엄격하게 제어하는 것이 무엇보다 중요합니다.

1.2 DFX 설계의 구성 요소: 주요 용어 해설

DFX 설계 흐름과 문제 해결 과정을 정확하게 이해하기 위해서는 다음과 같은 핵심 용어에 대한 명확한 정의가 필요합니다.

- 정적 영역**(Static Region):** 부분 재구성 과정 중에도 수정되지 않고 계속 활성 상태를 유지하는 설계의 일부입니다. 시스템의 핵심 기능, 제어 로직, 외부 인터페이스 등이 주로 여기에 위치합니다.[4]
- 재구성 가능 파티션**(Reconfigurable Partition, RP):** 동적 로직을 위해 미리 정의되고 물리적으로 할당된 FPGA 패브릭 내의 "슬롯" 또는 영역입니다. RP는 설계 계층 구조의 특정 블록에 정의됩니다.[4]
- 재구성 가능 모듈**(Reconfigurable Module, RM):** 해당 RP에 맞도록 컴파일된 특정 논리 기능 또는 모듈(예: FFT 코어, 암호화 엔진)입니다. 하나의 RP에 대해 여러 개의 다른 RM을 설계할 수 있으며, 런타임에 이들을 교체할 수 있습니다.[4]
- 파티션 핀**(Partition Pins):** 정적 로직과 RP 사이의 경계에 위치하는 물리적 연결 지점입니다. 이는 RM 포트의 물리적 구현체로서 라우팅 및 타이밍에 매우 중요하며, Vivado 툴이 자동으로 생성하는 앵커 포인트 역할을 합니다.[5]
- **Pblock:** RP를 구성하는 특정 FPGA 리소스(SLICE, BRAM, DSP 등)를 정의하는 물리적 제약 조건입니다. 이는 DFX 설계에서 사용자가 정의하는 가장 중요한 제약 조건 중 하나입니다.[2] 모든 RP는 반드시 Pblock이 할당되어야 하며, 그렇지 않으면 place_design 단계에서 HDPR-30 오류가 발생하며 실패합니다.[9]

1.3 Vivado DFX 흐름 탐색: 프로젝트 모드 vs. 비프로젝트 모드

Vivado Design Suite에서 DFX 설계를 진행하는 흐름은 일반적인 설계 흐름과 몇 가지 중요한 차이점이 있습니다.

- **DFX 활성화:** 기존 프로젝트를 DFX 프로젝트로 변환하려면 Tools > Enable Dynamic Function eXchange 메뉴를 선택해야 합니다. 이 작업은 되돌릴 수 없으므로, 실행 전에 프로젝트를 아카이빙하는 것이 매우 중요한 모범 사례입니다.[2]
- 파티션 정의**:** RP는 계층적 모듈 인스턴스를 마우스 오른쪽 버튼으로 클릭하고 "Create Partition Definition"을 선택하여 생성합니다. 이후의 RM 추가 및 관리는 DFX Wizard를 통해 이루어집니다.[2]
- **DFX Wizard:** 이 GUI 도구는 DFX 프로젝트 구조를 관리하는 중심점입니다. "Configuration"(각 RP에 대해 하나의 RM을 포함하는 완전한 디자인 이미지)을 정의하고 RM을 RP와 연결하는 데 사용됩니다.[2]
- 비프로젝트**(Non-Project) Tcl** 기반 흐름**:** 자동화와 재현성을 위해 Tcl 스크립트 기반 흐름이 강력히 권장됩니다. 이 흐름은 합성(synth_design), 체크포인트

관리(read_checkpoint, write_checkpoint), 구현(opt_design, place_design, route_design), 비트스트림 생성(write_bitstream) 등 일련의 Tcl 명령어로 구성됩니다.[2]

1.4 DFX 전용 IP의 역할: 안전망 구축

AMD는 DFX 설계의 안정성과 디버깅 용이성을 높이기 위해 다음과 같은 핵심 IP를 제공합니다. 이러한 IP를 올바르게 사용하는 것은 견고한 DFX 시스템을 구축하는 데 필수적입니다.

- **DFX Controller (PG374):** 특히 SoC가 아닌 FPGA 전용 설계에서 재구성 프로세스를 실시간으로 관리하기 위한 IP 코어입니다. 하드웨어 또는 소프트웨어 트리거를 처리하고 메모리에서 부분 비트스트림 로딩을 관리할 수 있습니다.[3]
- **DFX Decoupler (PG375):** RP 경계를 가로지르는 신호를 논리적으로 격리하는 데 필수적입니다. 재구성 과정에서 RP 내부의 로직 값이 정의되지 않은 상태(X)가 정적 설계로 전파되는 것을 방지하여 시스템 오작동을 막습니다.[3]
- **DFX AXI Shutdown Manager:** AXI 인터페이스를 위한 특수 디커플러입니다. 재구성이 시작되기 전에 AXI 트랜잭션이 정상적으로 처리되고 인터페이스가 정지 상태(quiescent)가 되도록 보장하여 시스템 중단이나 데이터 손상을 방지합니다.[3]
- **DFX Bitstream Monitor (PG376):** 부분 비트스트림이 소스(예: DDR 메모리)에서 구성 엔진(예: ICAP)으로 전송되는 과정을 추적할 수 있는 강력한 디버깅 IP입니다. 이를 통해 재구성을 시도하기 전에 비트스트림 손상이나 전송 오류를 식별할 수 있습니다.[3]

DFX Decoupler와 AXI Shutdown Manager는 런타임 실패를 예방하기 위한 사전 예방적 도구이며, 견고한 설계 방법론의 일부입니다. 반면, DFX Bitstream Monitor는 문제가 발생했을 때 디버깅을 위한 사후 대응적 도구입니다. 초보 설계자들이 리소스를 절약하기 위해 디커플러를 생략하고 시스템 소프트웨어가 RP를 정지시킬 수 있다고 가정하는 것은 흔한 실수입니다. 이는 매우 취약한 접근 방식이며, 이러한 IP가 제공하는 하드웨어 기반 격리는 신뢰할 수 있는 DFX 시스템의 초석입니다. 따라서 런타임 중단 문제에 대한 트러블슈팅은 항상 이러한 IP의 올바른 구현 및 사용 여부를 확인하는 것에서 시작해야 합니다.

섹션 2: 사전 예방적 트러블슈팅: 모범 사례 및 제약 전략

이 섹션은 예방 가이드의 핵심입니다. 여기에 설명된 설계 관행과 제약 방법론을 올바르게 따르면 일반적인 DFX 구현 및 런타임 오류의 대부분을 방지할 수 있습니다.

2.1 플로어플래닝 마스터하기**: Pblock**의 기술과 과학

성공적인 DFX 설계의 가장 중요한 요소는 물리적 플로어플래닝, 즉 Pblock을 정확하게 정의하는 것입니다.

- **Pblock** 생성**:** 각 RP에 대해 Pblock을 생성해야 합니다. 이는 Vivado IDE의 Device View에서 그래픽으로 그리거나 Tcl 명령어(create_pblock, resize_pblock)를 사용하여 정의할 수 있습니다.[2] Pblock은 해당 RP에 배치될 가장 큰 RM이 요구하는 모든 리소스(SLICE, DSP, BRAM 등)를 포함해야 합니다.[9]
- 경계 규칙 및 정렬**:** Pblock 경계는 임의로 설정할 수 없습니다. 클럭 영역(Clock Region)과 같은 FPGA의 특정 아키텍처 구조에 맞춰 정렬되어야 합니다. 이를 준수하지 않는 것이 DRC(Design Rule Check) 위반의 주요 원인입니다.[22] 특히 7 시리즈 디바이스에서는 RESET_AFTER_RECONFIG 속성을 활성화하려면 Pblock이 클럭 영역과 수직으로 정렬되어야 합니다.[2]
- **SNAPPING_MODE** 속성**:** 이 속성은 Pblock 경계를 DRC 규정에 맞는 합법적인(legal) 경계로 자동 조정하는 강력한 도구입니다. 이 값을 ON 또는 ROUTING으로 설정하는 것은 UltraScale+ 및 Versal 디바이스에서 특히 권장됩니다.[2] 이 속성은 HDPR-25 및 HDPR-26과 같은 오류를 예방할 수 있습니다.
- 분할된**(Disjoint) Pblock:** 일반적으로 피해야 하지만, 때로는 분할된 Pblock이 필요할 수 있습니다. 그러나 이는 라우팅 문제를 야기할 수 있으므로 신중하게 사용해야 합니다.[9]
- 정적 로직 플로어플래닝**:** 정적 로직에 대해서도 Pblock을 생성하는 것이 유용할 수 있습니다. 특히 정적 넷이 RP 영역으로 "새어 들어가(bleed over)" 라우팅 경합을 일으키는 것을 방지하는 데 효과적입니다.[9]

SNAPPING_MODE는 자동 수정 기능으로 제시되지만 [22], 의도치 않은 결과를 초래할 수 있습니다. 정렬 규칙을 충족시키기 위해 Pblock을 확장하면서 인접한 RP나 정적 로직을 위해 예약된 리소스를 예기치 않게 침범할 수 있기 때문입니다. 이는 Versal 설계에서

SNAPPING_MODE가 추가 IRI_QUAD 타일을 끌어와 인접한 Pblock과 겹치게 만들어 HDPR-25 오류를 유발하는 경우에서 명확히 드러납니다.[24] 이 경우 최종 해결책은

resize_pblock -remove 명령어를 사용한 수동 재정의입니다. 이는 자동화된 도우미가 근본적인 물리적 아키텍처에 대한 이해를 대체할 수 없음을 보여줍니다. 견고한 설계 흐름은 먼저 SNAPPING_MODE를 사용한 다음, 결과적인 Pblock의 영역을 검증하고 과도하게 확장된 부분을 수동으로 수정하는 과정을 포함할 수 있습니다.

2.2 DFX 시스템에서 타이밍 클로저 달성하기

DFX 설계의 타이밍 클로저는 정적 영역과 동적 영역 간의 경계를 신중하게 관리하는 데 달려 있습니다.

- 클럭 네트워크 설계**:** RP 경계를 가로지르는 클럭("경계 클럭")은 특별한 주의가 필요합니다. 클럭 소스는 가급적 정적 영역에 위치시키고, 전용 클럭 버퍼를 통해 RP로 공급하는 것이 이상적입니다.[27] MMCM/PLL과 같은 클럭 수정 블록을 RP 내부에 배치하는 것은 가능하지만, 상당한 복잡성을 추가하며 엄격한 플로어플래닝 규칙을 따라야 합니다.[9]
- 경계 경로 제약**:** 정적/동적 경계를 가로지르는 모든 신호는 적절하게 제약되어야 합니다. 여기에는 RM의 모든 입력과 출력을 레지스터링하여 깨끗하고 동기화된 인터페이스를 만드는 것이 포함됩니다.[9] 이는 타이밍 분석을 단순화하고 정적 및 동적 영역 내의 타이밍 경로를 격리시킵니다.
- 클럭 스큐 관리**:** 정적 영역과 동적 영역 간 경로의 높은 클럭 스큐는 타이밍 실패의 일반적인 원인입니다. 클럭 소스 및 통신하는 정적 로직과 관련된 RP의 신중한 플로어플래닝이 중요합니다.[28] 여러 RP를 구동하는 정적 레지스터를 복제하는 것도 경로 균형을 맞추고 팬아웃을 줄이는 데 도움이 될 수 있습니다.[9]
- 클럭 도메인 교차**(CDC):** DFX 경계에서의 모든 비동기 CDC는 검증된 표준 동기화 회로를 사용하여 극도의 주의를 기울여 처리해야 합니다. DFX Decoupler IP는 재구성 중에 이러한 교차가 안전하게 유지되도록 돕습니다.[3]

2.3 인터페이스 무결성 및 디커플링

- 디커플링의 필요성**:** 재구성 중에는 RP 내부의 로직이 정의되지 않은 상태에 있습니다. RP에서 정적 영역으로 향하는 모든 신호는 무작위로 토글될 수 있으며, 이는 정적 로직에 치명적인 오류를 유발할 수 있습니다. 따라서 이러한 신호를 알려진 안전한 상태로 유지하기 위한 디커플링 로직은 필수적입니다.[3]
- 리셋 전략**:** 견고한 리셋 전략은 매우 중요합니다. Pblock 속성인 RESET_AFTER_RECONFIG는 부분 비트스트림이 로드된 후 RM의 내부 레지스터에 하드웨어 기반 리셋을 제공하여 깨끗한 시작 상태를 보장합니다.[2] 이는 애플리케이션 레벨의 리셋과는 별개입니다.
- **AXI** 인터페이스 관리**:** SoC 설계에서는 AXI 인터페이스가 경계에서 흔히 사용됩니다. DFX AXI Shutdown Manager IP는 재구성 전후 및 도중에 이러한 복잡한 인터페이스가 적절히 정지되도록 보장하여 교착 상태나 트랜잭션 오류를 방지하는 권장 모범 사례입니다.[3]

2.4 디바이스 제품군별 **DFX** 가이드라인 비교

DFX 구현 규칙과 기능은 디바이스 세대에 따라 크게 발전했습니다. 7 시리즈 프로젝트에서

Versal 프로젝트로 전환하는 설계자는 완전히 새로운 제약 조건과 기능(예: NoC, 기본 흐름으로서의 블록 디자인 컨테이너)에 직면하게 됩니다. 다음 표는 이러한 차이점을 한눈에 요약하여, 오래되거나 잘못된 가정을 적용하여 발생하는 주요 오류 원인을 방지합니다.

기능/제약 조건	7-Series 디바이스	UltraScale/UltraScale+ 디바이스	Versal 디바이스
주요 설계 흐름	비프로젝트 Tcl 흐름 권장	프로젝트 흐름 지원 강화	블록 디자인 컨테이너(BDC) 기반 프로젝트 흐름이 주류
RESET_AFTER_RECONFIG	Pblock 속성, 클럭 영역 정렬 필요 [2]	기본적으로 항상 활성화됨 [2]	기본적으로 항상 활성화됨
클리어링 비트스트림	사용 불가	일부 흐름에서 필요 (_partial_clear.bit) [2]	일부 흐름에서 필요
클럭킹 규칙	엄격한 글로벌 클럭킹 규칙 [5]	유연성 증가, 새로운 제약 추가	MBUFGCE 분주 출력 제한 등 새로운 제약 존재 [9]
고속 트랜시버	특정 플로어플래닝 규칙 적용 [9]	특정 플로어플래닝 규칙 적용 [9]	특정 플로어플래닝 규칙 적용
NoC(Network on Chip)	해당 없음	해당 없음	특정 소유권 및 애퍼처 규칙과 함께 지원 [9]
추상 셸**(Abstract Shell)** 흐름	지원 안 됨	지원됨 [11]	지원됨 [3]

섹션 **3:** 사후 대응적 트러블슈팅**:** 일반적인 오류 및 해결책 카탈로그

이 섹션은 가이드의 "응급실"에 해당합니다. 구현 중에 발생하는 일반적인 오류 메시지와 실패

시나리오에 대한 구체적이고 실행 가능한 해결책을 제공합니다. 먼저, 문제 발생 시 빠른 진단을 돕기 위한 참조표를 제시합니다.

증상	가장 가능성 높은 원인	관련 섹션
opt_design 또는 place_design 중 DRC 오류 발생	Pblock/플로어플랜 위반	3.1
route_design 후 타이밍 위반 (Negative Slack)	혼잡, 부적절한 플로어플랜, 제약되지 않은 경계 경로	3.2, 2.2
링커 오류 (Undefined HARPData)	프로젝트/넷리스트 손상	3.2
하드웨어에서 부분 비트스트림 로딩 시 CRC 오류 보고	비트스트림 손상 또는 잘못된 디바이스 구성 모드	3.3
재구성은 성공했으나 새로운 모듈이 동작하지 않음	RM이 리셋 상태에 머무름, 디커플링 문제, 소프트웨어 버그	3.3, 4.2

3.1 주요 DRC 위반(HDPR 코드) 심층 분석

이 하위 섹션에서는 가장 일반적인 HDPR(Hierarchical Design / Partial Reconfiguration) DRC에 대한 상세한 분석을 제공합니다.

- **HDPR-25: Reconfigurable Pblocks must not overlap.**
  - 증상: [24]에서 볼 수 있는 오류 메시지.
  - 원인: 두 개의 Pblock이 동일한 FPGA 타일을 포함하도록 물리적으로 정의되었습니다. 2.1절에서 분석한 바와 같이, Versal에서 흔한 원인은 SNAPPING_MODE 속성이 돌출된 IRI_QUAD 사이트를 추가하여 인접한 Pblock을 서로 침범하도록 자동 확장하는 경우입니다.
  - 해결책: 겹치는 리소스를 제거하도록 Pblock을 수동으로 편집해야 합니다. Tcl 명령어 resize_pblock <pblock_name> -remove [get_sites {<site_pattern>}]를 사용합니다. [26] 및 [24]에 명시된 예시는 다음과 같습니다:
    resize_pblock pblock_rp1rm1 -remove.

- **HDPR-30: Missing PBLOCK On Reconfigurable Cell.**
  - 증상**:** [10] 및 [9]에서 볼 수 있는 오류 메시지.
  - 원인**:** 이는 근본적인 위반입니다. 인스턴스가 재구성 가능하도록 표시되었지만(HD.RECONFIGURABLE 속성이 true로 설정됨), 해당 인스턴스와 연관된 Pblock이 생성되지 않았습니다. [10]에서 확인된 특정 코너 케이스는 Versal Classic SoC Boot 흐름에서 구현을 실행하기 전에 합성된 디자인을 열면 이 속성이 누락될 수 있다는 것입니다.
  - 해결책**:** Pblock이 생성되고 재구성 가능한 셀과 올바르게 연결되었는지 확인합니다. [10]의 코너 케이스에 대한 해결 방법은 합성된 디자인을 닫고, 실행을 리셋한 다음, 합성과 구현을 함께 다시 시작하는 것입니다.
- **HDPR-50 / HDPR-18: Programmable Unit missing range / No Pblock range for cell.**
  - 증상**:** 사용자가 리소스를 추가했다고 생각함에도 불구하고 Pblock 내에 특정 리소스 유형(예: BUFGCE)이 누락되었다는 오류 메시지.[30]
  - 원인**:** 이는 미묘한 문제입니다. 프로그래밍 가능 단위(Programmable Unit, PU)는 물리적으로 관련된 여러 사이트 유형의 그룹입니다. Pblock이 유효하려면 특정 PU에 필요한 모든 사이트 유형을 포함해야 합니다. 하나라도 누락되면 SNAPPING_MODE가 사용자가 명시적으로 추가한 사이트를 포함하여 전체 PU를 제거할 수 있으며, 이로 인해 이 오류가 발생합니다.
  - 해결책**:** 사용자는 사용하려는 PU에 대해 Pblock 범위가 포괄적인지 확인해야 합니다. 이를 위해서는 디바이스 아키텍처에 대한 더 깊은 이해가 필요합니다. 해결 방법은 해당 PU에 대해 누락된 사이트 유형을 식별하고 resize_pblock 명령어에 추가하는 것입니다.
- **HDPR-87: Clock Net Rule Violation.**
  - 증상**:** RM 내에서 소싱된 클럭 넷이 하나 이상의 파티션 핀을 통해 RP 외부의 로드를 구동하는 것으로 발견됨.[27]
  - 원인**:** RM 내부에서 생성되거나 버퍼링된 클럭이 RM 내에서 분기되어 여러 로드를 구동하고, 이 경로들이 서로 다른 물리적 위치에서 RP를 빠져나갑니다. DFX 클럭킹 규칙은 클럭 넷이 단일하고 예측 가능한 지점에서 경계를 넘도록 요구합니다.
  - 해결책**:** 클럭 분기 로직을 재구성 가능 모듈(RM)에서 정적 영역으로 이동해야 합니다. 클럭은 단일 파티션 핀을 통해 RP를 빠져나간 다음, 정적 로직에서 버퍼링/분기되어 다양한 목적지로 구동되어야 합니다. [27]의 한 사용자는 인터페이스 번들 대신 클럭 핀을 개별적으로 연결하여 문제를 해결했는데, 이는 툴 최적화 문제를 시사합니다.
- 기타 **HDPR** 및 배치 오류**:** 이 섹션에서는 [37] 및 [53]의 HDPR-59(부적절한 클럭 로드), Place 30-730(넷 연결 해제 불가), PLSTAT-4(부적절하게 배치된 사이트), HDPR-6(RP Pblock에 부적절하게 배치된 로직) [36] 등 다른 오류들도 다룰 것입니다. 각 오류는 원인과 해결책과 함께 상세히 설명됩니다.

3.2 합성 및 기타 구현 실패

- [Common 17-70] Application Exception: Undefined HARPData for the netlist [37]:
  종종 프로젝트 또는 넷리스트 데이터베이스의 손상을 나타냅니다. 해결책은 실행을 리셋하고 재빌드하거나, 심각한 경우 Tcl 스크립트를 사용하여 프로젝트를 재생성하는 것입니다.
- 다중 리소스 할당 / 다중 드라이버 [38]:
  일반적인 VHDL/Verilog 오류이지만, 신호가 정적 로직과 (잘못된) 재구성 가능 로직 양쪽에서 구동될 경우 DFX에서 특히 위험합니다. 엄격한 인터페이스 정의가 핵심입니다.
- 타이밍 실패 **(Setup/Hold** 위반**):** 구현 후 타이밍이 실패하면, 첫 번째 단계는 임계 경로 보고서(critical path report)를 분석하는 것입니다.[28] 실패가 정적 로직 내에 있는지, RM 내에 있는지, 아니면 경계를 가로지르는 경로에 있는지 확인해야 합니다. 경계 경로 실패는 플로어플래닝 문제나 인터페이스 레지스터 누락을 가리킵니다. RM 내에서의 실패는 Pblock이 너무 작거나, 모양이 이상하거나, 디바이스의 리소스가 부족한 영역에 위치하여 라우팅 혼잡을 유발했음을 나타낼 수 있습니다.[33]

3.3 비트스트림 및 하드웨어 구성 실패

- 부분 비트스트림 로딩 중 CRC 오류 [42]:
  부분 재구성 중에 하드웨어(예: ICAP을 통해)에서 보고된 CRC 오류는 문제의 강력한 지표입니다.
  - 원인 **1:** 비트스트림 손상**.** 비트 파일이 저장 또는 전송 중에 손상되었을 수 있습니다. DFX Bitstream Monitor IP는 이 문제를 디버깅하는 데 이상적인 도구입니다.[3]
  - 원인 **2:** 잘못된 구성 모드**.** [42]에 자세히 설명된 바와 같이, SSI(다중 다이) 디바이스의 경우 모드 핀을 통해 "JTAG Only" 구성 모드를 사용하면 마스터 ICAP이 슬레이브 SLR에 액세스하는 것을 제한합니다. 이로 인해 부분 재구성이 실패합니다. JTAG는 모드 핀과 상관없이 항상 사용 가능하므로 해결책은 모드 핀을 변경하는 것입니다.
  - 원인 **3:** 잘못된 비트스트림 생성 옵션**.** [43]의 한 사용자는 프로세서 로딩을 위해 .bit를 .bin 파일로 변환할 때 write_cfgmem 명령어에서 -disablebitswap 옵션을 생략하여 발생한 ICAP 로딩 문제를 해결하는 데 2주를 소비했습니다. 이는 비트스트림 변환에 정확하고 올바른 툴 명령어를 사용하는 것이 얼마나 중요한지를 강조합니다.
- 재구성이 시작되지 않거나 중단됨 [43]:
  DFX Controller는 성공(상태 0x7 또는 0x107)을 보고하지만 RM이 변경되거나 작동하지 않습니다.
  - 원인**:** 이는 비트스트림 무결성 문제보다는 시스템 레벨의 로직 문제를 가리킵니다. RM이 리셋 상태에 머물러 있습니까? 디커플러 로직이 신호 통과를 막고 있습니까? 프로세서가 DFX Controller에게 비트스트림이 위치한 메모리의 올바른 주소를 정확히 가리키고 있습니까?
  - 해결책**:** ILA를 사용하여 DFX Controller의 AXI 인터페이스를 모니터링하여 전송되는 주소와 데이터를 확인합니다. 또한 RP로 들어가는 제어 신호(리셋, 인에이블)를 프로빙하여 재구성 후 RM이 작동하도록 허용되는지 확인합니다.

섹션 **4:** 고급 하드웨어 런타임 디버깅 기법

이 섹션은 "빌드는 되었지만 작동하지 않는" 문제, 즉 트러블슈팅에서 가장 어려운 단계를 다룹니다.

4.1 디버그 코어**(ILA/VIO)**를 이용한 **DFX** 설계 계측

- 과제**:** DFX 설계를 디버깅하려면 정적 영역과 활성 RM 모두에 대한 가시성이 필요합니다. 디버그 코어(ILA, VIO)를 배치하는 것은 독특한 과제를 제시합니다: 정적 로직에 배치해야 할까요, 아니면 동적 로직에 배치해야 할까요?.[44]
- 디버그 허브 및 코어 배치**:** Vivado 툴은 RM 내부에 ILA를 배치하는 것을 지원합니다. 이 흐름은 특수 S_BSCAN_ 포트를 통해 RM의 디버그 코어와 정적 영역의 주 디버그 허브 간의 연결을 자동으로 추론합니다.[22] 그러나 이는 취약할 수 있습니다. 지원 포럼의 한 사용자[52]는 툴이 디버그 연결을 비활성화하는 LUT를 잘못 삽입하여 수동 제약 조건으로 수정해야 했다고 보고했습니다.
- 모범 사례**:** 정적/동적 경계의 신호를 디버깅할 때는 ILA를 정적 영역에 인스턴스화하고 경계 넷을 거기서 프로빙하는 것이 더 견고합니다. 이렇게 하면 디버그 로직이 항상 존재하며 재구성에 영향을 받지 않습니다. ILA는 경계에서 볼 수 없는 RM 내부 깊숙한 로직을 관찰해야 할 경우에만 RM 내부에 배치해야 합니다.
- 통합 분석**:** [44]은 디버그 코어 사용에 대한 일반적인 정보를 제공합니다. [22] 및 [52]은 DFX 관련 세부 사항과 잠재적인 함정에 대한 중요한 정보를 제공합니다.

4.2 정적**/**동적 경계를 넘나드는 디버깅 전략

- 재구성 이벤트 트리거링**:** 정적 영역의 VIO 코어를 사용하여 이벤트를 수동으로 트리거하거나 상태를 모니터링할 수 있습니다. 예를 들어, VIO 출력이 DFX Controller를 트리거하여 재구성을 시작하게 할 수 있습니다. 그런 다음 ILA를 이 VIO 신호에 트리거하도록 설정하여 교체 직전, 도중, 직후의 시스템 상태를 캡처할 수 있습니다.
- 정적 및 동적 동작의 상관관계 분석**:** ILA가 정적 영역에 있을 때, RP로 들어가는 입력과 나오는 출력을 모니터링할 수 있습니다. 입력은 올바른데 출력이 잘못되었다면 문제는 RM 내부에 있습니다. 정적 로직의 입력이 잘못되었다면 문제는 다른 곳에 있습니다. 이는 디버깅을 위한 필수적인 논리적 격리를 제공합니다.
- **JTAG-to-AXI** 마스터 사용**:** SoC 설계의 경우 JTAG-to-AXI 마스터 디버그 코어는 매우

유용합니다. 이를 통해 프로세서를 일시 중지하고 Vivado Hardware Manager에서 DFX Controller의 레지스터나 정적 로직 내의 상태 레지스터를 포함한 모든 AXI 주변 장치에 수동으로 읽기/쓰기를 수행할 수 있습니다.[47] 이는 하드웨어를 탓하기 전에 소프트웨어가 시스템을 올바르게 구성했는지 확인하는 데 사용할 수 있습니다.

DFX 디버깅에서 흔히 발생하는 혼란스러운 사용자 오류 중 하나는 .ltx 파일 불일치 문제입니다. Vivado Hardware Manager는 디자인의 논리적 넷 이름을 물리적 디버그 코어 데이터에 매핑하기 위해 디버그 프로브 파일(.ltx)을 사용합니다. DFX 설계에는 여러 구성이 존재합니다. 만약 FPGA를 config_1으로 프로그래밍하고 config_2용으로 생성된 .ltx 파일을 사용하여 연결을 시도하면, Hardware Manager는 프로브 불일치를 보고하거나 연결에 실패할 것입니다.[48]
견고한 디버그 방법론은 어떤

.bit/.pdi 파일과 .ltx 파일이 어떤 하드웨어 구성에 해당하는지 세심하게 관리해야 합니다. 이 가이드에서는 사용자에게 이 문제에 대해 명시적으로 경고하고 DFX 디버그 결과물에 대한 명확한 파일 이름 지정 및 관리 규칙을 권장해야 합니다.

섹션 **5:** 탄력적인 **DFX** 흐름을 위한 자동화 및 스크립팅

이 섹션에서는 섹션 2의 모범 사례를 코드화하여 보다 신뢰할 수 있고 오류 발생 가능성이 적은 빌드 프로세스를 만드는 실용적인 Tcl 스크립팅 예제를 제공합니다.

5.1 Tcl 기반 **DFX** 프로젝트 관리 모범 사례

- 소스 제어**:** 핵심 모범 사례는 버전 제어 시스템에 저장된 소스 파일로부터 프로젝트를 재생성하는 Tcl 스크립트를 사용하는 것입니다. 이는 재현성을 보장하고 Vivado 프로젝트 파일에 포함된 절대 경로 문제를 방지합니다.[49]
  rebuild.tcl과 같은 스크립트는 모든 HDL 소스, IP 소스(.xci), 제약 조건 파일(.xdc)을 읽어 새로운 프로젝트를 생성해야 합니다.
- 모듈식 스크립트**:** 빌드 프로세스를 모듈식 Tcl 스크립트로 분할합니다: 프로젝트 생성용, 합성용, 각 구성 구현용 등으로 나눕니다.[17] 이렇게 하면 흐름을 더 쉽게 관리하고 디버깅할 수 있습니다. Kria DFX GitHub 저장소[51]는 이러한 모듈식 구조의 훌륭한 예를 제공합니다.

5.2 필수 **Tcl** 스크립트 및 스니펫

- 프로젝트 생성 및 **DFX** 활성화**:**
  Tcl
  # 프로젝트 생성 및 소스 파일 추가
  create_project my_dfx_proj./my_dfx_proj -part <part_name>
  add_files {./rtl/static_top.v./rtl/rm1.v...}
  add_files -fileset constrs_1 {./constraints/top.xdc}

  # 중요: 이 작업은 되돌릴 수 없습니다!
  set_property PR_FLOW 1 [current_project]

- **Pblock** 및 물리적 제약 조건 적용**:**
  Tcl
  # 모범 사례: 물리적 제약은 전용 XDC 또는 Tcl 스크립트에서 적용
  create_pblock pblock_rp1
  add_cells_to_pblock [get_pblocks pblock_rp1][get_cells {inst_rp1}]
  resize_pblock [get_pblocks pblock_rp1] -add {SLICE_X...:SLICE_Y...}
  set_property SNAPPING_MODE ON [get_pblocks pblock_rp1]
  set_property RESET_AFTER_RECONFIG true [get_pblocks pblock_rp1]

  2

- 부모 및 자식 구성 구현**:** 스크립트는 먼저 부모 구현(impl_1)을 시작하고, 정적 라우팅을 잠근 다음, 잠긴 정적 체크포인트를 가져와 다른 RM에 대한 후속 자식 구현을 시작합니다.
- 검증 스크립트 **(pr_verify):** 모든 구성이 구현된 후에는 pr_verify Tcl 명령을 실행해야 합니다. 이 중요한 단계는 정적 디자인과 모든 RM 간의 라우팅 충돌을 확인하여, 어떤 RM의 라우팅도 정적 로직을 불법적으로 방해하지 않도록 보장합니다. 이는 비트스트림을 생성하기 전의 최종 온전성 검사입니다.[6]

섹션 **6:** 결론 및 전문가 권장 사항

이 마지막 섹션에서는 핵심 내용을 요약하고 설계자가 DFX 구현을 시작하기 전에 검토할 수 있는 상위 수준의 체크리스트를 제공합니다.

6.1 DFX 트러블슈팅 방법론 요약

- 물리적으로 생각하기**:** DFX 문제의 대부분은 물리적 구현 문제입니다. 올바른 플로어플래닝과 경계 설계를 최우선으로 고려해야 합니다.
- 사전 예방적으로 제약하기**:** 설계 주기 초기에 Pblock, 타이밍 및 인터페이스에 대한

명확하고 견고한 제약 조건을 정의해야 합니다. 툴이 추측하게 두어서는 안 됩니다.
- 격리 및 디커플링: RP 경계에는 항상 디커플러 로직을 사용해야 합니다. 이는 신뢰할 수 있는 시스템을 위해 선택 사항이 아닙니다.
- 모든 것을 자동화하기: Tcl 스크립트를 사용하여 빌드가 반복 가능하고 검증 가능하며 수동 오류 발생 가능성이 적도록 보장해야 합니다.
- 체계적으로 디버깅하기: 런타임 오류가 발생하면 ILA 및 기타 디버그 IP를 사용하여 문제를 소프트웨어, 정적 하드웨어 또는 특정 재구성된 모듈로 논리적으로 격리해야 합니다.

6.2 구현 전 DFX 설계 검토 체크리스트

프로젝트 리더와 설계자를 위한 최종적이고 간결한 체크리스트입니다.

- [ ] DFX 흐름을 활성화하기 전에 DFX 프로젝트가 제대로 아카이빙되었는가?
- [ ] 모든 RP가 Pblock과 연결되어 있는가?
- [ ] Pblock 경계가 아키텍처 정렬 규칙에 맞게 검토되었는가? SNAPPING_MODE가 활성화되었는가?
- [ ] RP 경계를 가로지르는 모든 신호가 양쪽에서 레지스터링되었는가?
- [ ] RP에서 정적으로 향하는 모든 신호에 디커플링 로직(예: DFX Decoupler IP)이 인스턴스화되었는가?
- [ ] RESET_AFTER_RECONFIG 사용을 포함한 명확한 리셋 전략이 정의되었는가?
- [ ] 모든 경계 클럭이 DFX 클럭킹 규칙을 준수하는지 검토되었는가?
- [ ] 구현된 모든 구성에 대해 pr_verify가 실행되었는가?

참고 자료

1. 文件类型: User Guides - 查看所有版本, 10월 2, 2025에 액세스, https://china.xilinx.com/support/documentation-navigation/see-all-versions.html?xlnxproducttypes=Design%20Tools&xlnxdocumentid=UG909#!
2. How-To on Partial Reconfiguration with Vivado - 01signal.com, 10월 2, 2025에 액세스, https://www.01signal.com/vendor-specific/xilinx/partial-reconfiguration/part2-vivado-flow/
3. Dynamic Function eXchange (DFX) - AMD, 10월 2, 2025에 액세스, https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/dynamic-function-exchange.html
4. Dynamic Function eXchange (DFX) — Kria SOM DFX Examples 1.0 documentation - GitHub Pages, 10월 2, 2025에 액세스, https://xilinx.github.io/kria-apps-docs/dfx.html
5. Vivado Design Suite User Guide: Partial Reconfiguration (UG909) - ivPCL, 10월 2, 2025에 액세스, http://ivpcl.unm.edu/ivpclpages/Research/drastic/PRWebPage/ug909-vivado-part

ial-reconfiguration.pdf
6. Partial Reconfiguration - CERN Indico, 10월 2, 2025에 액세스, https://indico.cern.ch/event/1020703/contributions/4284100/attachments/2213972/3747775/PartialReconfiguration.pdf
7. Dynamic Function eXchange - Technology Blogs, 10월 2, 2025에 액세스, https://blog.abbey1.org.uk/index.php/technology/dynamic-function-exchange
8. Vivado Design Suite Tutorial: Partial Reconfiguration (UG947), 10월 2, 2025에 액세스, http://dllamocca.org/Tutorials/EmbSys/Unit_6/ug947-vivado-partial-reconfiguration-tutorial.pdf
9. Implementing the DFX Design - 2025.1 English - UG909, 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Implementing-the-DFX-Design
10. Vivado 2021.2 - DRC HDPR-30 Classic SoC Missing PBLOCK On Reconfigurable Cell, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/article/Vivado-HDPR-30-Classic-SoC-Missing-PBLOCK
11. Step 3: Setting Up the Design for DFX - 2024.1 English - UG947, 10월 2, 2025에 액세스, https://docs.amd.com/r/2024.1-English/ug947-vivado-partial-reconfiguration-tutorial/Step-3-Setting-Up-the-Design-for-DFX
12. Creating a Dynamic Function eXchange Project - 2025.1 English - UG909, 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Creating-a-Dynamic-Function-eXchange-Project
13. Isolation Design Flow + Dynamic Function eXchange Example Application Note, 10월 2, 2025에 액세스, https://fpga.eetrend.com/files/2021-07/wen_zhang_/100114232-211414-gechishejiliuchengdongtaihanshujiaohuanshili.pdf
14. Vivado Design Suite Tutorial: Partial Reconfiguration (UG947) - ivPCL, 10월 2, 2025에 액세스, http://ivpcl.unm.edu/ivpclpages/Research/drastic/PRWebPage/ug947-vivado-partial-reconfiguration-tutorial.pdf
15. Vivado Tcl Build Script - Project F, 10월 2, 2025에 액세스, https://projectf.io/posts/vivado-tcl-build-script/
16. TCL script Vivado Project Tutorial - Surf-VHDL, 10월 2, 2025에 액세스, https://surf-vhdl.com/tcl-script-vivado-project-tutorial/
17. hdlguy/vivado_tcl: demo project to show how to use vivado tcl scripts to do everything. - GitHub, 10월 2, 2025에 액세스, https://github.com/hdlguy/vivado_tcl
18. Step 2: Customizing the Dynamic Function eXchange (DFX) Controller IP - 2025.1 English, 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug947-vivado-partial-reconfiguration-tutorial/Step-2-Customizing-the-Dynamic-Function-eXchange-DFX-Controller-IP
19. FPGA Dynamic Function eXchange - Webthesis, 10월 2, 2025에 액세스, https://webthesis.biblio.polito.it/16011/1/tesi.pdf

20. Downloading the bitstream freezes everything - Support - PYNQ, 10월 2, 2025에 액세스, https://discuss.pynq.io/t/downloading-the-bitstream-freezes-everything/5204
21. DFX(Dynamic Function eXchange) - AMD, 10월 2, 2025에 액세스, https://www.amd.com/ko/products/adaptive-socs-and-fpgas/technologies/dynamic-function-exchange.html
22. Vivado Design Suite Tutorial: Dynamic Function eXchange, 10월 2, 2025에 액세스, https://static.eetrend.com/files/2021-12/wen_zhang_/100556482-232279-ug947-vivado-partial-reconfiguration-tutorial.pdf
23. Vivado Design Suite Tutorial - ivPCL, 10월 2, 2025에 액세스, http://ivpcl.unm.edu/ivpclpages/Research/drastic/PRWebPage/Vivado%20Design%20Suite%20Tutorial%20using%20ZedBoard.pdf
24. 000035182 - Vivado DFX 2023.1 - HDPR-25 DRC related to ..., 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/article/000035182
25. Debugging Versal Device DFX Designs - 2025.1 English - UG909, 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Debugging-Versal-Device-DFX-Designs
26. 000035182 - Vivado DFX 2023.1 - HDPR-25 DRC related to protruding IRI_QUAD sites, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/article/000035182?language=zh_CN
27. ERROR: [DRC HDPR-87] during place_design for DFX, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/question/0D54U00005uBDUVSA4/error-drc-hdpr87-during-placedesign-for-dfx?language=en_US
28. Timing Closure Techniques - Designing with Xilinx FPGAs Using Vivado - FPGAkey, 10월 2, 2025에 액세스, https://www.fpgakey.com/tutorial/section838
29. Timing Closure - 2025.1 English - UG949, 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug949-vivado-design-methodology/Timing-Closure
30. 64176 - Vivado UltraScale Partial Reconfiguration - DRC (HDPR-50) still occurs even if all BUFGCE/MMCM_ADV ranges in the clock range are added into Reconfigurable Module's pblock - Adaptive Support, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/article/64176
31. Design Considerations - 2025.1 English - UG909, 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Design-Considerations
32. Nested DFX Design Considerations - 2025.1 English - UG909, 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Nested-DFX-Design-Considerations
33. Overcoming Timing Closure Challenges in FPGA Projects - RunTime Recruitment, 10월 2, 2025에 액세스, https://runtimerec.com/overcoming-timing-closure-challenges-in-fpga-projects/
34. Advice: Summary of source of timing issues and ways to fix in FPGA? - Reddit, 10월 2, 2025에 액세스,

https://www.reddit.com/r/FPGA/comments/ud8kwy/advice_summary_of_source_of_timing_issues_and/
35. Design Considerations and Guidelines for Versal Devices - 2025.1 English - UG909, 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Design-Considerations-and-Guidelines-for-Versal-Devices
36. 000038192 - ERROR: [DRC HDPR-6] during place_design in Vivado 2025.1, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/article/000038192?
37. DFX - Adaptive Support - AMD, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/topic/0TO2E000000YO9hWAG/dfx?language=en_US
38. Top 10 Common VHDL Coding Errors and How to Fix Them - FPGATEK, 10월 2, 2025에 액세스, https://fpgatek.com/vhdl-coding-errors/
39. How to Ensure Your FPGA Design Meets Timing Closure | by Ampheo | Medium, 10월 2, 2025에 액세스, https://medium.com/@pqshedy33/how-to-ensure-your-fpga-design-meets-timing-closure-4fea8537cab1
40. Interpreting Common Xilinx Compilation Errors in the LabVIEW FPGA Module - NI, 10월 2, 2025에 액세스, https://www.ni.com/en/support/documentation/supplemental/18/interpreting-common-xilinx-compilation-errors-in-the-labview-fpg.html
41. How to Avoid Common FPGA Programming Errors - RunTime Recruitment, 10월 2, 2025에 액세스, https://runtimerec.com/how-to-avoid-common-fpga-programming-errors/
42. 000034268 - Vivado DFX - loading of Partial bitfile fails on SSI device - Adaptive Support, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/article/000034268
43. DFX Controller using ICAP not loading new RM - Adaptive Support - AMD, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/question/0D52E00006iHp0WSAS/dfx-controller-using-icap-not-loading-new-rm?language=en_US
44. Debugging Dynamic Function eXchange (DFX) Designs in Vivado ..., 10월 2, 2025에 액세스, https://docs.amd.com/r/en-US/ug908-vivado-programming-debugging/Debugging-Dynamic-Function-eXchange-DFX-Designs-in-Vivado-Hardware-Manager
45. Hardware Debugging | FPGA Design with Vivado, 10월 2, 2025에 액세스, https://xilinx.github.io/xup_fpga_vivado_flow/lab6.html
46. Advanced Debug Techniques for Hardware Engineers - BLT Inc., 10월 2, 2025에 액세스, https://bltinc.com/xilinx-training-courses/advanced-debug-techniques-for-hardware-engineers/
47. Interacting with Debug Instrumentation - Designing with Xilinx FPGAs Using Vivado, 10월 2, 2025에 액세스, https://www.fpgakey.com/tutorial/section771
48. Vivado VIO debug issue : r/FPGA - Reddit, 10월 2, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/1da8aro/vivado_vio_debug_issue/

49. jhallen/vivado_setup: How to set up Xilinx Vivado for source control - GitHub, 10월 2, 2025에 액세스, https://github.com/jhallen/vivado_setup
50. Vivado Development Best Practices : r/FPGA - Reddit, 10월 2, 2025에 액세스, https://www.reddit.com/r/FPGA/comments/4yyo78/vivado_development_best_practices/
51. Xilinx/kria-dfx-hw - GitHub, 10월 2, 2025에 액세스, https://github.com/Xilinx/kria-dfx-hw
52. Synthesis Issue with Vivado Debug Cores inside Dynamic Function eXchange Reconfigurable Partition - Adaptive Support, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/question/0D54U00007FyztsSAB/synthesis-issue-with-vivado-debug-cores-inside-dynamic-function-exchange-reconfigurable-partition?language=en_US
53. 000035406 - Dynamic Function eXchange (DFX) - Known issues, 10월 2, 2025에 액세스, https://adaptivesupport.amd.com/s/article/000035406