ZCU670 기반 적응형 SDR: DFX를 이용한 다중 대역 채널 필터 동적 구현 가이드

I. 서론: 차세대 SDR을 위한 하드웨어 적응성

현대의 무선 통신 시스템은 LTE, 5G NR, Wi-Fi 등 다양한 표준이 공존하며, 각기 다른 주파수 대역과 대역폭 요구사항을 가집니다. 소프트웨어 정의 라디오(SDR)는 이러한 복잡성에 대응하기 위한 핵심 기술이지만, 프로그래머블 로직(PL)에 모든 잠재적 신호 처리 체인을 구현하는 것은 리소스 낭비와 전력 소모 증가로 이어집니다. AMD의 Dynamic Function eXchange(DFX)는 이러한 한계를 극복하고, 시스템 동작 중에 PL의 특정 영역을 용도에 맞게 동적으로 재구성하여 진정한 의미의 하드웨어 적응성을 구현하는 강력한 솔루션입니다.[1]

본 튜토리얼은 고급 Zynq UltraScale+ MPSoC가 탑재된 AMD ZCU670 평가 보드를 사용하여, DFX 기술의 실제적인 적용 사례를 단계별로 안내합니다. 우리는 8채널 디지털 채널 필터 뱅크를 구현하되, LTE 대역용 필터 세트와 5G NR 3.5GHz 대역용 필터 세트를 별도의 재구성 모듈(RM)로 설계합니다. 그 후, 시스템의 운영 모드에 따라 PetaLinux 환경에서 실행되는 소프트웨어가 적절한 필터 모듈을 PL에 동적으로 로드하여 하드웨어 기능을 실시간으로 전환하는 과정을 시연합니다.

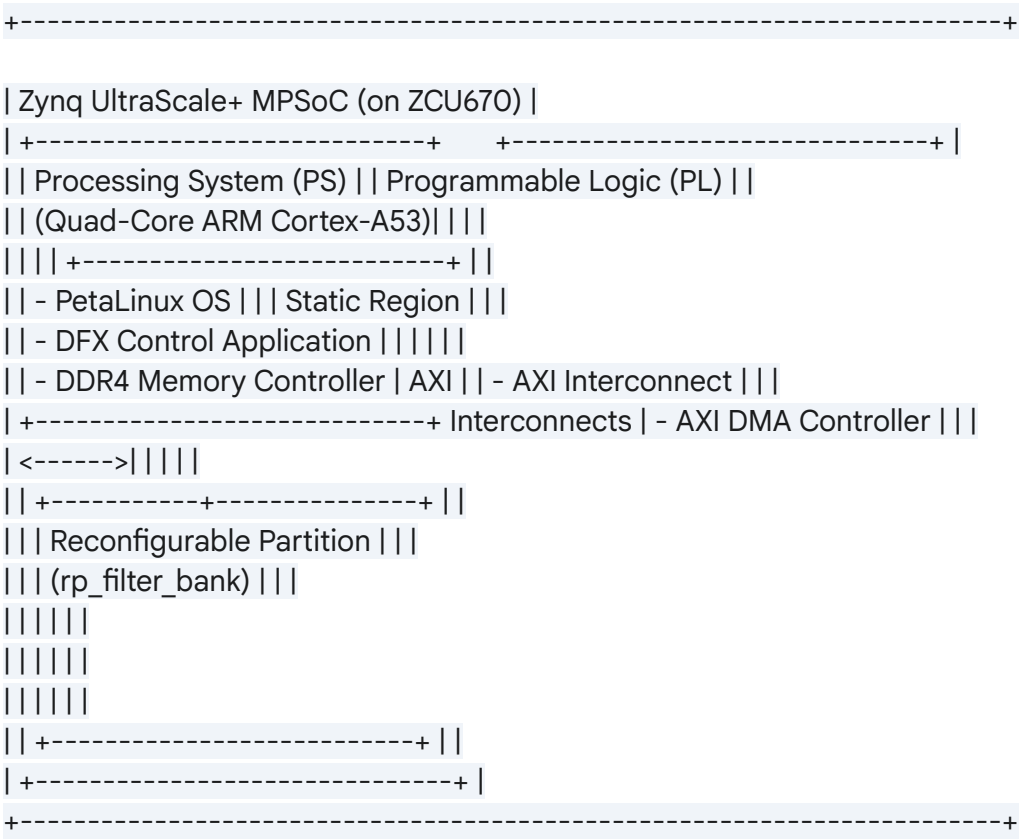
1.1. 프로젝트 목표 및 아키텍처 개요

본 튜토리얼의 최종 목표는 다음과 같습니다.

- 정적(Static) SDR 플랫폼 구축: Zynq UltraScale+ MPSoC의 처리 시스템(PS)과 데이터 전송을 위한 핵심 AXI 인프라를 포함하는 안정적인 기반 시스템을 설계합니다.
- 재구성 가능한 필터 슬롯(Slot) 정의: PL 내에 8채널 필터 뱅크가 자리할 재구성 파티션(RP)을 정의합니다.
- 두 가지 필터 모듈(RM) 개발:
  1. rm_lte_filter_bank: LTE 채널 대역폭에 최적화된 8개의 FIR 필터 세트.
  2. rm_5g_filter_bank: 5G NR 3.5GHz 대역의 더 넓은 채널 대역폭에 최적화된 8개의 FIR 필터 세트.

- 소프트웨어 제어 시스템 구현**:** PetaLinux 운영 체제에서 간단한 쉘 스크립트를 통해 사용자가 원하는 필터 모듈을 선택하고 PL에 로드하는 DFX 제어 메커니즘을 구축합니다.

그림 **1:** 목표 시스템 아키텍처



II. 1부: Vivado를 이용한 하드웨어 플랫폼 설계

본 섹션에서는 Vivado Design Suite를 사용하여 DFX 하드웨어 플랫폼을 처음부터 구축하는 과정을 상세히 안내합니다.

2.1. 프로젝트 생성 및 정적 시스템 구성

1. **Vivado** 실행 및 프로젝트 생성**:** Vivado를 실행하고 Create Project를 클릭합니다. 프로젝트 이름(예: zcu670_dfx_filter)을 지정하고 RTL Project를 선택합니다.
2. 보드 선택**:** Boards 탭에서 ZCU670 Evaluation Board를 검색하여 선택합니다. 보드 파일이 설치되어 있지 않다면, Vivado 설치 시 추가하거나 AMD 웹사이트에서 다운로드해야 합니다.
3. 블록 디자인 생성**:** Flow Navigator에서 IP INTEGRATOR -> Create Block Design을 클릭합니다. 디자인 이름(예: top_design)을 지정합니다.
4. **Zynq MPSoC IP** 추가**:** 다이어그램 창에서 + 버튼을 클릭하여 Zynq UltraScale+ MPSoC IP를 추가합니다.
5. 기본 구성 적용**:** IP가 추가되면 상단에 나타나는 녹색 배너에서 Run Block Automation을 클릭합니다. 나타나는 대화상자에서 Apply Board Preset이 선택된 것을 확인하고 OK를 클릭합니다. 이 과정은 ZCU670 보드에 맞게 DDR 컨트롤러, 클럭, I/O 등의 PS 설정을 자동으로 구성합니다.
6. **PL** 클럭 및 리셋 활성화**:** Zynq MPSoC 블록을 더블 클릭하여 Re-customize IP 창을 엽니다. Clock Configuration -> Output Clocks -> PL Fabric Clocks로 이동하여 PL_CLK0를 활성화하고 원하는 주파수(예: 100MHz)를 설정합니다. PS-PL Configuration -> PS-PL Interfaces로 이동하여 Master Interface 아래의 M_AXI_HPM0_FPD와 Slave Interface 아래의 S_AXI_HP0_FPD를 활성화합니다.
7. **AXI DMA** 추가**:** + 버튼을 클릭하여 AXI Direct Memory Access IP를 추가합니다. 이 IP는 PS의 DDR 메모리와 PL의 스트리밍 인터페이스 간 데이터 전송을 담당합니다.
8. 연결 자동화 실행**:** Run Connection Automation을 클릭합니다. Vivado가 AXI DMA의 AXI-Lite 제어 인터페이스(S_AXI_LITE)를 PS의 M_AXI_HPM0_FPD에, 그리고 DMA의 메모리 맵 마스터 인터페이스(M_AXI_SG, M_AXI_MM2S, M_AXI_S2MM)를 PS의 S_AXI_HP0_FPD에 연결하도록 제안할 것입니다. 모든 항목을 선택하고 OK를 클릭합니다. 또한, axi_aclk를 pl_clk0에 연결하도록 선택합니다.
9. 디자인 검증**:** Validate Design (F6 키)을 실행하여 현재까지의 정적 시스템에 오류가 없는지 확인합니다.

2.2. 재구성 파티션**(RP)** 정의

1. **DFX** 활성화**:** Tools -> Enable Dynamic Function eXchange를 선택하여 DFX 워크플로우를 시작합니다.
2. **RP** 생성**:** DFX Wizard가 나타나면 Convert 탭을 선택하고 Instantiate를 클릭합니다. Cell name에 rp_filter_bank를 입력하고 OK를 클릭하여 재구성 파티션으로 사용할 계층 블록을 생성합니다.
3. **RP** 인터페이스 정의**:** 생성된 rp_filter_bank 블록을 선택하고 마우스 오른쪽 버튼을 클릭하여 Add Pin을 선택합니다. 다음 핀들을 추가하여 모든 RM이 공유할 표준 '소켓' 인터페이스를 정의합니다.
   - s_axis_data (Input, Type: axis)

- m_axis_data (Output, Type: axis)
- ap_clk (Input, Type: clk)
- ap_rst_n (Input, Type: rst)

4. **RP** 연결**:**
   - rp_filter_bank의 ap_clk 핀을 Zynq MPSoC의 pl_clk0에 연결합니다.
   - Processor System Reset IP의 peripheral_aresetn 출력을 rp_filter_bank의 ap_rst_n 핀에 연결합니다.
   - AXI DMA의 M_AXIS_MM2S 포트를 rp_filter_bank의 s_axis_data 포트에 연결합니다.
   - rp_filter_bank의 m_axis_data 포트를 AXI DMA의 S_AXIS_S2MM 포트에 연결합니다. 이로써 PS DDR -> DMA -> 필터(RP) -> DMA -> PS DDR의 데이터 경로가 완성됩니다.

2.3. 재구성 모듈**(RM)** 생성

이제 RP에 로드될 두 가지 다른 필터 뱅크를 별도의 블록 디자인으로 생성합니다.

1. **RM 1: LTE** 필터 뱅크 생성
   - File -> IP -> Create Block Design을 선택하여 새 블록 디자인을 생성하고 이름을 rm_lte_filter_bank로 지정합니다.
   - 이 블록 디자인 내부에 AXI4-Stream Broadcaster IP 1개, FIR Compiler IP 8개, AXI4-Stream Combiner IP 1개를 추가합니다.
   - AXI4-Stream Broadcaster의 Number of Master Interfaces를 8로 설정합니다.
   - 8개의 FIR Compiler IP를 각각 LTE 채널 대역폭에 맞는 계수(coefficients)로 설정합니다. (튜토리얼의 편의를 위해, 여기서는 기본 저역 통과 필터 계수를 사용합니다.)
   - AXI4-Stream Combiner의 Number of Slave Interfaces를 8로 설정합니다.
   - 입력 포트(s_axis_data)를 Broadcaster에 연결하고, Broadcaster의 8개 출력을 각 FIR 필터에, 각 FIR 필터의 출력을 Combiner의 8개 입력에, 그리고 Combiner의 출력을 출력 포트(m_axis_data)에 연결합니다. 모든 클럭과 리셋을 연결합니다.
   - 검증용 **ID** 레지스터 추가**:** AXI GPIO IP를 추가하고, All Outputs로 설정, GPIO Width를 32로 설정합니다. AXI-Lite 인터페이스를 외부로 노출시키고(Make External), GPIO 출력 포트를 Constant IP에 연결하여 고유값(예: 0x1111LTEF)을 설정합니다. 이 AXI-Lite 인터페이스는 나중에 정적 영역의 AXI Interconnect에 연결되어 PS가 접근할 수 있게 됩니다.
2. **RM 2: 5G NR** 필터 뱅크 생성
   - 위와 동일한 과정으로 rm_5g_filter_bank라는 새 블록 디자인을 생성합니다.
   - 내부 구조는 동일하지만, 8개의 FIR Compiler IP를 5G NR의 더 넓은 대역폭에 맞는 다른 계수로 설정합니다.
   - 검증용 AXI GPIO의 Constant IP 값을 다른 고유값(예: 0x5555GNRF)으로 설정합니다.
3. **DFX Wizard**에서 **RM** 할당**:**
   - top_design 블록 디자인으로 돌아와 DFX Wizard를 다시 엽니다 (Tools -> Dynamic

Function eXchange Wizard).
    - Add Reconfigurable Module을 클릭하여 rm_lte_filter_bank와 rm_5g_filter_bank를 추가합니다.
    - Configuration 탭에서 두 개의 구성을 생성합니다. config_1은 rm_lte_filter_bank를 사용하고, config_2는 rm_5g_filter_bank를 사용하도록 설정합니다.

2.4. 구현 및 비트스트림 생성

1. **Floorplanning:** DFX Wizard의 Floorplan 탭으로 이동하여 rp_filter_bank 파티션에 대한 Pblock을 생성합니다. Draw Pblock을 사용하여 디바이스 뷰에서 물리적 영역을 할당합니다. 이 영역은 두 RM 중 더 큰 것을 수용할 수 있을 만큼 충분해야 합니다.[3]
2. 구현 실행**:** Design Runs 창에서 impl_1 (config_1용)을 마우스 오른쪽 버튼으로 클릭하고 Launch Runs를 선택합니다. Vivado는 먼저 정적 로직과 rm_lte_filter_bank를 포함하는 부모 실행(parent run)을 구현합니다. 완료되면, 정적 로직의 결과를 고정한 채 rm_5g_filter_bank에 대한 자식 실행(child run)을 자동으로 수행합니다.
3. 비트스트림 생성**:** 모든 구현이 성공적으로 완료되면, Tcl 콘솔에서 write_bitstream -all 명령을 실행합니다. 이 명령은 config_1에 대한 전체 비트스트림과 각 RM에 대한 부분 비트스트림(rm_lte_filter_bank.bit, rm_5g_filter_bank.bit)을 생성합니다.
4. 하드웨어 내보내기**:** File -> Export -> Export Hardware를 선택합니다. Output을 Include bitstream으로 설정하고, 파일 이름(예: zcu670_dfx_wrapper.xsa)을 지정하여 PetaLinux에서 사용할 하드웨어 플랫폼 파일을 내보냅니다.

III. 2부**: PetaLinux** 임베디드 시스템 빌드

이제 Vivado에서 생성한 하드웨어 플랫폼을 기반으로 임베디드 Linux 시스템을 구축합니다.

3.1. PetaLinux 프로젝트 생성 및 구성

1. 프로젝트 생성**:** 터미널에서 PetaLinux 환경을 설정한 후 다음 명령을 실행합니다.
   ```
   Bash
   petalinux-create --type project --template zynqMP --name zcu670_dfx_os
   cd zcu670_dfx_os
   ```

2. 하드웨어 구성 가져오기**:** Vivado에서 내보낸 XSA 파일을 사용하여 PetaLinux 프로젝트를

구성합니다.

```
Bash
petalinux-config --get-hw-description=<path_to_xsa_file>/zcu670_dfx_wrapper.xsa
```

구성 메뉴가 나타나면 별다른 변경 없이 저장하고 종료합니다.

3.2. 루트 파일 시스템에 펌웨어 추가

1. 커널 구성 확인**:** petalinux-config -c kernel을 실행하고 Device Drivers -> FPGA Configuration Framework에서 Xilinx ZynqMP FPGA Manager가 활성화되어 있는지 확인합니다 (보통 기본값으로 활성화됨).
2. 펌웨어 파일 준비**:** Vivado 프로젝트의 impl_1/ 및 child_0_impl_1/ 디렉토리에서 생성된 부분 비트스트림 파일(*_partial.bit)을 찾습니다. 이 파일들을 *_partial.bin으로 변환해야 합니다.

```
Bash
# Vivado Tcl Console에서 실행
write_cfgmem -format BIN -interface SMAPx32 -loadbit "up 0x0
rm_lte_filter_bank_partial.bit" rm_lte.bin
write_cfgmem -format BIN -interface SMAPx32 -loadbit "up 0x0
rm_5g_filter_bank_partial.bit" rm_5g.bin
```

3. 레시피 수정**:** 생성된 rm_lte.bin과 rm_5g.bin 파일을 루트 파일 시스템에 포함시키기 위해 <proj_dir>/project-spec/meta-user/recipes-core/images/petalinux-image-append.bb 파일을 열고 다음 내용을 추가합니다.

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

SRC_URI += " \
    file://rm_lte.bin \
    file://rm_5g.bin \
"

do_install:append() {
    install -d ${D}/lib/firmware
    install -m 0644 ${WORKDIR}/rm_lte.bin ${D}/lib/firmware/
    install -m 0644 ${WORKDIR}/rm_5g.bin ${D}/lib/firmware/
}
```

4. 파일 복사**:** <proj_dir>/project-spec/meta-user/recipes-core/images/ 디렉토리 아래에 files라는 새 디렉토리를 만들고, 이 files 디렉토리에 rm_lte.bin과 rm_5g.bin 파일을

복사합니다.

3.3. 시스템 빌드 및 패키징

1. 빌드: 프로젝트 최상위 디렉토리에서 다음 명령을 실행하여 전체 시스템을 빌드합니다.
   ```
   Bash
   petalinux-build
   ```

2. 부팅 이미지 생성: SD 카드 부팅에 필요한 파일들을 패키징합니다.
   ```
   Bash
   petalinux-package --boot --fsbl images/linux/zynqmp_fsbl.elf --fpga
   images/linux/system.bit --u-boot --force
   ```

   이제 images/linux/ 디렉토리에 BOOT.BIN, image.ub, boot.scr 파일이 생성됩니다.

IV. 3부: 배포 및 동적 기능 검증

4.1. 보드 부팅

1. FAT32로 포맷된 Micro SD 카드에 images/linux/ 디렉토리의 BOOT.BIN, image.ub, boot.scr 파일을 복사합니다.
2. ZCU670 보드의 부팅 모드 스위치를 SD 카드 부팅으로 설정하고 SD 카드를 삽입합니다.
3. USB-UART 케이블을 연결하여 PC와 시리얼 콘솔을 설정하고(Baud rate: 115200), 보드에 전원을 공급합니다. PetaLinux 부팅 과정을 콘솔에서 확인하고 petalinux 사용자로 로그인합니다.

4.2. DFX 동적 로딩 및 검증

로그인 후, FPGA Manager의 sysfs 인터페이스를 사용하여 PL을 동적으로 재구성하고, 각 RM에 포함된 검증용 ID 레지스터를 읽어 기능 전환을 확인합니다.

1. **LTE** 필터 뱅크 로드 및 검증**:**
    - 터미널에 다음 명령어를 입력하여 LTE 필터 모듈을 로드합니다.
      Bash
      ```
      # FPGA Manager를 부분 재구성 모드로 설정
      echo 1 > /sys/class/fpga_manager/fpga0/flags

      # LTE 필터 뱅크 펌웨어 로드
      echo rm_lte.bin > /sys/class/fpga_manager/fpga0/firmware
      echo "LTE Filter Bank Loaded."
      ```
    - 이제 devmem 유틸리티를 사용하여 LTE 모듈의 ID 레지스터 값을 읽습니다. (참고: AXI GPIO의 주소는 Vivado의 Address Editor에서 확인해야 하며, 여기서는 0xA0000000으로 가정합니다.)
      Bash
      ```
      # AXI GPIO 주소에서 32비트 값을 읽음
      devmem 0xA0000000 32
      ```
    - 예상 결과**:** 터미널에 0x1111LTEF가 출력되어야 합니다.
2. **5G NR** 필터 뱅크 로드 및 검증**:**
    - 이번에는 5G NR 필터 모듈을 로드합니다.
      Bash
      ```
      # 5G NR 필터 뱅크 펌웨어 로드
      echo rm_5g.bin > /sys/class/fpga_manager/fpga0/firmware
      echo "5G NR Filter Bank Loaded."
      ```
    - 다시 devmem을 사용하여 ID 레지스터 값을 읽습니다.
      Bash
      ```
      devmem 0xA0000000 32
      ```
    - 예상 결과**:** 터미널에 0x5555GNRF가 출력되어야 합니다.

이 두 단계의 결과는 Linux 사용자 공간에서 실행된 간단한 명령어를 통해 PL의 하드웨어 기능이 성공적으로, 그리고 예측 가능하게 동적으로 교체되었음을 명확하게 증명합니다.

표 **1: DFX** 제어 및 검증을 위한 핵심 명령어 요약

명령어	목적
echo 1 > /sys/class/fpga_manager/fpga0/flags	FPGA Manager를 부분 재구성 모드로 설정합니다.
echo <firmware_name.bin> >	지정된 부분 비트스트림을 PL의 재구성

/sys/class/fpga_manager/fpga0/firmware	파티션에 로드합니다.
devmem <address> 32	지정된 물리 메모리 주소의 값을 읽어 현재 로드된 하드웨어의 상태를 확인합니다.

V. 결론

본 튜토리얼은 AMD ZCU670 보드에서 Dynamic Function eXchange를 활용하여 SDR 애플리케이션의 핵심 구성 요소인 디지털 필터 뱅크를 동적으로 교체하는 전체 과정을 시연했습니다. 이 접근 방식을 통해 단일 하드웨어 플랫폼이 LTE 및 5G NR과 같은 여러 통신 표준의 요구사항에 실시간으로 적응할 수 있음을 확인했습니다.

DFX 기술은 단순히 리소스를 절약하는 것을 넘어, 현장에서의 원격 하드웨어 업그레이드, 임무에 따른 기능 변경, 시스템의 전력 프로파일 최적화 등 SDR 시스템 설계에 새로운 차원의 유연성을 부여합니다.[1] 본 튜토리얼에서 다룬 원리와 기법들을 기반으로, 개발자들은 더욱 복잡하고 지능적인 적응형 무선 시스템을 구축해 나갈 수 있을 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. 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
3. Dynamic Function eXchange - Technology Blogs, 9월 27, 2025에 액세스, https://blog.abbey1.org.uk/index.php/technology/dynamic-function-exchange
4. Introduction - 2025.1 English - UG909, 9월 27, 2025에 액세스, https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration/Introduction
5. 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