ADALM-PLUTO용 애플리케이션 크로스 컴파일 상세 보고서**: sysroot** 활용

1.0 서론**: PlutoSDR** 온보드**(On-board)** 개발의 필요성

ADALM-PLUTO는 내부에 독립적인 ARM Cortex-A9 프로세서와 임베디드 리눅스 운영체제를 탑재하고 있어, 단순한 PC 주변기기를 넘어 독립적인 연산 장치로서의 잠재력을 지니고 있습니다.[1] 호스트 PC에서

libiio를 통해 원격으로 PlutoSDR을 제어하는 것이 일반적인 사용 방식이지만, 실시간성이 중요하거나 독립형(standalone)으로 동작해야 하는 애플리케이션(예: 현장 데이터 로거, 독립형 스펙트럼 모니터, 맞춤형 변조 신호 송신기 등)을 구현하기 위해서는 PlutoSDR의 ARM 프로세서에서 직접 실행되는 네이티브 바이너리를 개발해야 합니다.

이러한 개발 방식을 **크로스 컴파일(Cross-compilation)**이라고 합니다. 즉, 개발용 호스트 PC(일반적으로 x86-64 아키텍처)에서 PlutoSDR의 ARM 아키텍처용 실행 파일을 생성하는 과정을 의미합니다. 이 과정의 핵심은 단순히 다른 아키텍처용 코드를 생성하는 것을 넘어, 대상 시스템(PlutoSDR)의 라이브러리 및 헤더 파일 환경을 호스트 PC에 정확히 복제하여 링크(link) 문제를 해결하는 것입니다. 이 복제된 환경을 **sysroot**라고 부릅니다.

본 보고서는 ADALM-PLUTO 펌웨어 개발 환경을 기반으로 sysroot를 설정하고, 이를 활용하여 libiio와 같은 핵심 라이브러리를 PlutoSDR용으로 크로스 컴파일하는 전 과정을 기술적으로 상세히 설명합니다.

2.0 크로스 컴파일 생태계의 구성 요소

성공적인 크로스 컴파일 환경은 세 가지 핵심 요소로 구성됩니다: ARM 교차 컴파일러, 대상 시스템의 sysroot, 그리고 빌드 시스템(예: CMake, Make).

2.1 ARM 교차 컴파일러 **(Toolchain)**

교차 컴파일러는 특정 CPU 아키텍처(이 경우 ARM)를 위한 기계어 코드를 생성하는 컴파일러, 링커, 그리고 기타 유틸리티(assembler, objcopy 등)의 집합입니다. PlutoSDR 펌웨어 빌드 시스템은 특정 버전의 툴체인과의 호환성을 요구합니다. 공식 plutosdr-fw 리포지토리는 Linaro에서 배포하는 gcc-linaro-7.3.1-2018.05 버전을 사용하도록 지정하고 있습니다.[3] 빌드 시스템은

CROSS_COMPILE 환경 변수를 통해 사용할 툴체인의 접두사(prefix)를 인식합니다. 예를 들어, CROSS_COMPILE이 arm-linux-gnueabihf-로 설정되면, 빌드 시스템은 arm-linux-gnueabihf-gcc, arm-linux-gnueabihf-ld 등의 명령어를 호출합니다.

2.2 sysroot의 개념과 역할

sysroot는 크로스 컴파일에서 가장 중요한 개념 중 하나입니다. 컴파일러가 소스 코드를 컴파일하고 링크할 때, #include <libxml/parser.h>와 같은 헤더 파일이나 -lxml2와 같은 라이브러리 링크 요청을 만나게 됩니다. 이때 호스트 PC의 /usr/include나 /usr/lib에 있는 파일들은 x86-64 아키텍처용이므로 ARM 바이너리를 만드는 데 사용할 수 없습니다.

sysroot는 대상 시스템(PlutoSDR)의 루트 파일 시스템(/)을 호스트 PC의 특정 디렉토리에 그대로 복사해 놓은 것입니다. 여기에는 대상 시스템의 C 라이브러리(libc.so), libiio, libxml2 등 모든 공유 라이브러리와 관련 헤더 파일들이 ARM 아키텍처에 맞는 버전으로 들어 있습니다. 컴파일러에 --sysroot=<path_to_sysroot> 옵션을 전달하면, 컴파일러는 시스템 헤더 파일과 라이브러리를 호스트 PC의 경로가 아닌 지정된 sysroot 경로 내에서 찾게 됩니다.

PlutoSDR의 경우, 이 sysroot는 plutosdr-fw 펌웨어 전체 빌드 과정에서 Buildroot에 의해 자동으로 생성됩니다.[3] 따라서 사용자 애플리케이션을 크로스 컴파일하기 위한 첫 번째 단계는 항상 공식 펌웨어를 최소 한 번 이상 빌드하여 이

sysroot 디렉토리를 생성하는 것입니다.

3.0 단계별 가이드**: libiio** 크로스 컴파일

이 섹션에서는 실제 예시로, PlutoSDR의 핵심 통신 라이브러리인 libiio를 호스트 PC에서 PlutoSDR용으로 크로스 컴파일하는 과정을 상세히 설명합니다.

3.1 사전 준비**: sysroot** 생성

1. **PlutoSDR** 펌웨어 소스 코드 복제**:** 아직 수행하지 않았다면, 공식 펌웨어 리포지토리를 재귀적으로 복제합니다.
   Bash
   ```
   git clone --recursive https://github.com/analogdevicesinc/plutosdr-fw.git
   cd plutosdr-fw
   ```
   3
2. 펌웨어 전체 빌드**:** 이전 보고서에서 설명한 개발 환경(Linaro 툴체인, Xilinx Vivado 등)이 설정된 상태에서 make 명령을 실행하여 전체 펌웨어를 빌드합니다.
   Bash
   ```
   make
   ```
   5

   이 과정은 수십 분 이상 소요될 수 있으며, 성공적으로 완료되면 plutosdr-fw/buildroot/output/host/arm-buildroot-linux-gnueabihf/sysroot/ 경로에 PlutoSDR의 sysroot가 생성됩니다. 이 디렉토리의 내용을 살펴보면 usr/lib, usr/include 등 친숙한 리눅스 파일 시스템 구조를 확인할 수 있습니다.

3.2 크로스 컴파일 환경 설정

libiio 소스 코드를 컴파일하기 전에, 셸 환경이 교차 컴파일러와 sysroot를 올바르게 가리키도록 설정해야 합니다. 다음 스크립트는 필요한 모든 환경 변수를 설정하는 예시입니다.

Bash

```
# 1. Linaro 툴체인 경로 설정 (사용자 환경에 맞게 수정)
export TOOLCHAIN_PATH=/opt/linaro/gcc-linaro-7.3.1-2018.05-i686_arm-linux-gnueabihf
export PATH=$TOOLCHAIN_PATH/bin:$PATH

# 2. CROSS_COMPILE 접두사 설정
export CROSS_COMPILE=arm-linux-gnueabihf-

# 3. SYSROOT 경로 설정 (plutosdr-fw 빌드 결과물)
```

```
export PLUTOFW_DIR=/path/to/your/plutosdr-fw
export SYSROOT=$PLUTOFW_DIR/buildroot/output/host/arm-buildroot-linux-gnueabihf/sysroot

# 4. 컴파일러 및 링커 플래그 설정
#    - CFLAGS: 컴파일러에 sysroot 내의 include 경로를 알려줌
#    - LDFLAGS: 링커에 sysroot 내의 lib 경로를 알려줌
export CFLAGS="--sysroot=$SYSROOT -I$SYSROOT/usr/include"
export LDFLAGS="--sysroot=$SYSROOT -L$SYSROOT/usr/lib -L$SYSROOT/lib"
export CXXFLAGS=$CFLAGS
```

3.3 libiio 소스 코드 컴파일

이제 libiio를 컴파일할 준비가 되었습니다. libiio는 cmake 빌드 시스템을 사용합니다.

1. **libiio** 소스 코드 복제**:**
   ```
   Bash
   git clone https://github.com/analogdevicesinc/libiio.git
   cd libiio
   mkdir build && cd build
   ```
   5
2. **CMake** 실행 및 크로스 컴파일 설정**:** cmake를 실행할 때, 우리가 크로스 컴파일을 하고 있다는 사실과 함께 컴파일러 및 sysroot 정보를 명시적으로 전달해야 합니다.
   ```
   Bash
   cmake../ \
    -DCMAKE_SYSTEM_NAME=Linux \
    -DCMAKE_SYSTEM_PROCESSOR=arm \
    -DCMAKE_C_COMPILER=${CROSS_COMPILE}gcc \
    -DCMAKE_CXX_COMPILER=${CROSS_COMPILE}g++ \
    -DCMAKE_FIND_ROOT_PATH=$SYSROOT \
    -DCMAKE_FIND_ROOT_PATH_MODE_PROGRAM=NEVER \
    -DCMAKE_FIND_ROOT_PATH_MODE_LIBRARY=ONLY \
    -DCMAKE_FIND_ROOT_PATH_MODE_INCLUDE=ONLY \
    -DENABLE_IPV6=OFF \
    -DENABLE_USB_BACKEND=ON \
    -DENABLE_NETWORK_BACKEND=ON
   ```
   - CMAKE_C_COMPILER: 사용할 C 컴파일러를 지정합니다.
   - CMAKE_FIND_ROOT_PATH: CMake에게 sysroot의 위치를 알려줍니다. find_package, find_library 등의 명령이 이 경로를 우선적으로 검색하게 됩니다.

3. 컴파일 및 설치**:**
   Bash
   make

   컴파일이 성공적으로 완료되면 build 디렉토리 내에 libiio.so 파일이 생성됩니다.

3.4 결과 검증

생성된 라이브러리 파일이 실제로 ARM용으로 컴파일되었는지 확인하는 것은 매우 중요합니다. file 유틸리티를 사용하여 아키텍처를 확인할 수 있습니다.

Bash

file libiio.so.1

정상적인 출력 예시:
libiio.so.1: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked,...
만약 출력이 x86-64를 포함한다면, 크로스 컴파일 설정이 잘못된 것이므로 환경 변수와 CMake 옵션을 다시 확인해야 합니다.

4.0 커스텀 바이너리의 펌웨어 통합

이렇게 생성된 커스텀 라이브러리나 실행 파일을 PlutoSDR 펌웨어에 포함시키는 방법은 두 가지가 있습니다.

1. 수동 복사 **(**테스트용**):** scp를 사용하여 실행 중인 PlutoSDR에 파일을 직접 복사할 수 있습니다. 이는 개발 및 디버깅 단계에서 빠르고 편리한 방법입니다.
   Bash
   scp my_custom_app root@192.168.2.1:/usr/local/bin/

2. **Buildroot** 패키지 통합 **(**배포용**):** 애플리케이션을 펌웨어의 영구적인 일부로 만들고 싶다면, Buildroot에 새로운 패키지를 추가하는 것이 가장 올바른 방법입니다. 이를 위해서는 plutosdr-fw/buildroot/package/ 디렉토리 아래에 자신의 애플리케이션을 위한 새로운 디렉토리를 만들고, 그 안에 Config.in 파일과 <mypackage>.mk 파일을 작성해야

합니다. 이 .mk 파일은 Buildroot에게 소스 코드를 어디서 가져와 어떻게 컴파일하고 최종 루트 파일 시스템의 어느 위치에 설치할지를 알려주는 Makefile 스크립트입니다.[4] 이 방식을 사용하면
make 한 번으로 모든 커스텀 코드가 포함된 최종 pluto.frm 펌웨어 이미지를 생성할 수 있습니다.

5.0 결론

ADALM-PLUTO용 애플리케이션을 크로스 컴파일하는 과정은 단순히 컴파일러를 바꾸는 것을 넘어, 대상 시스템의 파일 시스템 환경인 sysroot를 정확히 이해하고 활용하는 체계적인 엔지니어링 작업입니다. plutosdr-fw 빌드 시스템이 생성하는 sysroot는 PlutoSDR의 ARM 프로세서에서 실행될 고성능의 맞춤형 애플리케이션을 개발하기 위한 필수적인 기반을 제공합니다. 개발자는 이 보고서에 기술된 절차를 통해 libiio와 같은 기존 라이브러리를 컴파일하거나, 자신만의 독자적인 프로그램을 개발하여 PlutoSDR의 잠재력을 최대한 활용하는 독립형 SDR 시스템을 구축할 수 있습니다.