**CAFE77xxD 1.4.0**

AFE77xxD libraries

Sang-Gu Kang

1st – 2024.10.07

AFE77xxD 시리즈는 고성능 Analog Front End(AFE) 칩으로 RF 송수신 기능과 다양한 아날로그 처리 기능을 제공합니다. “인스턴스화”는 AFE77xxD 칩을 시스템 내에서 초기화하고 설정하여 실제로 작동하게 하는 과정을 의미합니다. AFE77xxD는 **AFE7705**, **AFE7706**, **AFE7769** 등 여러 모델로 구성될 수 있으며, 이들은 각각 다른 대역폭과 채널을 제공하지만, 설정 및 인스턴스화 과정은 기본적으로 유사합니다.

**AFE77xxD 인스턴스화 과정**

1. 전원 공급 및 초기화

* AFE77xxD 칩에 적절한 전원을 공급합니다. 칩은 다중 전원 레일을 요구하며, 각 전원 레일이 안정적으로 공급되어야 합니다.
* 칩에 클럭 신호를 입력합니다. AFE77xxD는 외부 클럭을 통해 동작하므로 시스템의 클럭 관리가 중요합니다.

2. SPI를 통한 설정

* AFE77xxD 칩은 SPI(Serial Peripheral Interface)를 통해 제어됩니다. 먼저 MCU나 FPGA와 AFE77xxD 사이에 SPI 통신을 설정해야 합니다.
* SPI 통신을 통해 레지스터 설정을 할 수 있습니다. 주요 레지스터 설정은 칩의 동작 모드, 출력 전력, 필터 설정 등을 정의하는 데 사용됩니다.

3. 클럭 및 동기화 설정

* JESD204B 또는 JESD204C 인터페이스를 사용할 경우, 프레임 클럭과 장치 간 동기화를 위해 정확한 클럭 설정이 필요합니다.
* RF 신호의 변환 및 처리를 위한 클럭 소스와 칩 내 PLL(Phase Locked Loop)을 구성합니다.

4. 디지털 인터페이스 설정 (JESD204)

* AFE77xxD 시리즈는 **JESD204B** 또는 **JESD204C** 인터페이스를 사용하여 디지털 데이터 변환기(ADC/DAC)와 FPGA/ASIC 간 데이터를 주고받습니다.
* JESD204 링크 파라미터를 설정합니다. 여기에는 **lane(레인) 수**, **데이터 속도**, **프레임 길이**, **멀티플렉싱 모드** 등이 포함됩니다.
* 링크가 성공적으로 설정되면, **SYNC** 신호를 통해 동기화 과정을 거칩니다.

5. RF 경로 설정

* RF 송수신 경로에 대한 설정을 진행합니다. 송신 경로에서 DAC의 설정과 출력 증폭기를 설정하고, 수신 경로에서는 ADC와 저잡음 증폭기(LNA) 등을 설정합니다.
* 각 채널의 이득(gain), 필터링 설정, 주파수 변환 및 기타 RF 파라미터를 설정합니다.

6. 구성된 설정을 테스트

* 설정 후에는 칩이 올바르게 동작하는지 확인하기 위해 테스트 절차를 진행합니다. 각 채널의 데이터가 정확하게 송수신되는지, JESD204 링크가 올바르게 설정되었는지 확인합니다.

7. 운영 모드 전환

* 설정이 완료된 후, AFE77xxD는 정상 동작 모드로 전환되어 데이터 송수신을 시작합니다. 필요 시 특정 주파수 대역을 전환하거나 운영 모드를 변경할 수 있습니다.

**설정 시 고려 사항**

* **전원 및 클럭 안정성**: AFE77xxD는 고속으로 동작하는 칩이기 때문에 전원 및 클럭 안정성이 매우 중요합니다. 불안정한 클럭 또는 전원이 공급될 경우 성능 저하나 오작동이 발생할 수 있습니다.
* **SPI 통신**: SPI를 통한 제어는 칩의 동작을 결정하므로, 적절한 통신 프로토콜을 설정하고 레지스터 맵에 맞는 값을 정확히 입력하는 것이 중요합니다.
* **RF 경로 최적화**: 시스템에서 요구하는 주파수 대역과 대역폭에 맞춰 AFE77xxD의 송수신 경로를 설정해야 합니다. 이때 필터링 및 이득 설정이 적절히 이루어져야 합니다.

**AFE77xxD 인스턴스화 예시 코드**

아래는 마이크로컨트롤러나 FPGA에서 AFE77xxD 칩을 초기화하고 설정하는 과정의 간단한 예시 코드입니다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

AFE77xxD 시리즈 칩의 인스턴스화 과정은 시스템 설계에 따라 달라질 수 있지만, 기본적으로는 전원 공급, SPI 통신 설정, JESD204 링크 구성, RF 경로 설정 등으로 이루어집니다.

목차

[ReadMe 1](#_Toc178961867)

[Introduction 1](#_Toc178961868)

[Key Features 2](#_Toc178961869)

[Folder Structure 2](#_Toc178961870)

[Instantiating the Libraries(라이브러리 인스턴스화) 2](#_Toc178961871)

[Version History 3](#_Toc178961872)

[Data Structures 3](#_Toc178961873)

[Files 3](#_Toc178961874)

# ReadMe

## Introduction

이러한 C 라이브러리는 AFE77xxD를 동적으로 제어하기 위한 API로 사용하도록 설계되었습니다. 이는 초기화 시퀀스를 처리하지 않고 장치의 초기화 후 동적 제어를 처리합니다. 라이브러리가 호스트 라이브러리에 쉽게 통합될 수 있도록 많은 유연성이 제공됩니다.

## Key Features

1. Error handling: 각 함수는 함수 실행이 성공했는지 여부를 반환합니다. 다음은 다양한 오류입니다.  
   a. TI\_AFE\_RET\_EXEC\_PASS: 함수가 성공적으로 실행되었습니다.  
   b. TI\_AFE\_RET\_EXEC\_FAIL: 매개변수가 유효하지만 실행이 실패한 경우.  
   c. TI\_AFE\_RET\_PARAM\_INVALID: 매개변수 반환이 잘못되었습니다.
2. Multi AFE devices support: 동일한 호스트에서 여러 장치를 제어하는 ​​데 대한 지원이 있습니다. afeInst는 모든 함수에 전달되는 매개변수입니다. 라이브러리에서 지원하는 이를 전달하는 방법에는 두 가지가 있으며, 이는 인스턴스화 섹션에서 설명합니다.
3. Channel Remapping: 시스템 관점에서 채널 번호가 AFE 관점에서와 다른 경우가 빈번합니다. 이 시나리오를 처리하기 위해 API 내에서 채널 번호를 리매핑하는 옵션이 있습니다.
4. Simulation Mode: 하드웨어 없이 소프트웨어 흐름 테스트를 가능하게 하기 위해 라이브러리는 시뮬레이션 모드를 지원합니다. 이는 setAfeLibsRunMode 함수를 사용하여 설정할 수 있습니다. 이 모드에서는 읽기 값이나 폴링 검사가 수행되지 않으며 함수 실행을 보장합니다.
5. Function Renaming: 모든 함수와 systemParams의 이름을 접두사로 지정하는 옵션이 있습니다. MakeFile에서 TI\_AFE77xxD\_FUNC\_NAME\_PREFIX를 정의하여 이를 수행할 수 있습니다.

## Folder Structure

1. AFE77xxD (Afe77xxD?): 이 폴더에는 사용자가 수정하지 않고도 사용할 수 있는 AFE 기능이 들어 있습니다.
2. Afe77xxDUser: 이 폴더에는 호스트가 정의해야 할 호스트 드라이버 기능과 설정이 들어 있습니다. 이 폴더의 주요 파일은 다음과 같습니다.  
   a. src\afeParameters1.c: 아래에 설명된 Method 1에서 사용되는 경우 afeInst structure 초기화를 위한 샘플 코드가 들어 있습니다. 값은 AFE configuration을 기반으로 인스턴스화해야 합니다.  
   b. src\afeParameters2.c: 아래에 설명된 Method 2에서 사용되는 경우 afeInst structure 초기화를 위한 샘플 코드가 들어 있습니다. 값은 AFE configuration을 기반으로 인스턴스화해야 합니다.  
   c. src\baseFunc.c: 여기에는 사용자가 작성해야 하는 호스트 드라이버 함수가 들어 있습니다. 이러한 함수 정의는 변경하면 안 됩니다.

## Instantiating the Libraries(라이브러리 인스턴스화)

이 섹션에서는 라이브러리를 인스턴스화하는 방법을 살펴보겠습니다.

NOTE 1: “인스턴스화”는 AFE77xxD 칩을 시스템 내에서 초기화하고 설정하여 실제로 작동하게 하는 과정을 의미.

호스트에서 제어하는 ​​각 AFE를 구분하기 위해 AFE 구성(샘플링 속도, 인터페이스 속도 및 JESD 매개변수와 같은 시스템 매개변수라고 함)과 채널 리매핑과 같은 기타 매개변수에 대한 정보가 **afe77DInstDeviceInfo** type structure로 그룹화됩니다. 이 구조는 호스트에서 제어하는 ​​각 AFE에 대해 한 번씩 인스턴스화해야 합니다. 각 함수에는 AFE에 대한 고유 식별자인 **afeInst** input argument가 있습니다. 이 동일한 afeInst는 모든 SPI, Logging, Sysref 드라이버 함수에도 전파됩니다. afe77DInstDeviceInfo에는 또한 void 포인터인 halConfig가 있으며, 이를 사용하여 구조를 확장하여 호스트 기능을 추가할 수 있습니다.

라이브러리를 인스턴스화하고 사용하는 데에는 두 가지 주요 접근 방식이 있습니다.

1. Method 1: 이 접근 방식에서는 afe77DInstDeviceInfo의 해당 포인터를 모든 함수에 전달해야 합니다.
2. Method 2: 이 접근 방식에서 afe77DInstDeviceInfo의 인스턴스는 tiAfe77D\_afeParameters.h에 표시된 대로 AFE당 tiAfe77DDeviceInfo\_t 배열로 정의되어야 하며 해당 AFE 인덱스는 함수에 전달되어야 합니다. 두 방법 모두 지원되는 기능에 차이가 없지만, 차이점은 해당 afe77DInstDeviceInfo 인스턴스를 함수에 전달하거나 AFE 인덱스를 전달하는 경우에만 있습니다.

Method 1을 사용하여 인스턴스화하는 경우: AFE77xxDUser\src\afeParameters1.c를 참조하세요. 예를 들어

1. afe77DInstDeviceInfo 구조의 배열을 tiAfe77D\_deviceInfo로 정의합니다. 인스턴스를 setDefaultParams 함수에 전달하여 값을 기본값으로 설정합니다. 필요에 따라 rxChannelRemap, txChannelRemap, fbChannelRemap, logLevel 및 halConfig의 값을 설정합니다. 나머지는 변경하지 마세요.
2. baseFunc.c에 드라이버 함수를 코딩합니다.
3. 함수 이름에 대한 사용자 정의 접두사가 필요한 경우 MakeFile에 적절한 플래그를 추가합니다. 이 플래그가 MakeFile에 추가되지 않으면 모든 함수에 대한 접두사로 "ti\_afe77D\_"가 추가됩니다.  
   예1: 접두사가 없는 경우 추가할 플래그:  
   -D'TI\_AFE77XXD\_FUNC\_NAME\_PREFIX(funcName)=(funcName)'  
   예2: 접두사로 "ti\_afe77D\_"를 추가할 플래그:  
   -D'TI\_AFE77XXD\_FUNC\_NAME\_PREFIX(funcName)=(ti\_afe77D\_##funcName)'
4. 채널 리매핑 기능을 원하지 않는 경우 컴파일 전에 -DENABLE\_RX\_CH\_REMAP, -DENABLE\_TX\_CH\_REMAP 및 -DENABLE\_FB\_CH\_REMAP 플래그 없이 컴파일합니다. 이렇게 하면 최적화를 위해 불필요한 로직이 컴파일되는 것을 방지할 수 있습니다. makefile에 -DUSE\_AFE77D\_LIB\_FORMAT\_2 플래그를 설정하지 마세요.
5. 채널 리매핑 기능을 사용하려면 afe77DInstDeviceInfo에서 rxChannelRemap, txChannelRemap, fbChannelRemap을 적절히 설정하세요.

Method 2를 사용하여 인스턴스화하는 경우: AFE77xxDUser\src\afeParameters2.c를 참조하세요. 예를 들어

1. 먼저 setNumAfe77D(NumAFEs) 함수를 사용하여 호스트가 제어하는 ​​AFE의 수를 설정합니다. 인스턴스를 setDefaultParams 함수에 전달하여 값을 기본값으로 설정합니다. 필요에 따라 rxChannelRemap, txChannelRemap, fbChannelRemap, logLevel 및 halConfig의 값을 설정합니다. 나머지는 변경하지 마세요. afe77DInstDeviceInfo 구조체의 배열을 tiAfe77D\_deviceInfo로 정의하고 해당 값으로 초기화합니다.
2. baseFunc.c에 드라이버 함수를 코딩합니다.
3. 함수 이름에 대한 사용자 정의 접두사가 필요한 경우 MakeFile에 적절한 플래그를 추가합니다. 이 플래그가 MakeFile에 추가되지 않으면 모든 함수에 대한 접두사로 "ti\_afe77D\_"가 추가됩니다.  
   예1: 접두사가 없는 경우 추가할 플래그:  
   -D'TI\_AFE77XXD\_FUNC\_NAME\_PREFIX(funcName)=(funcName)'  
   예2: 접두사로 "ti\_afe77D\_"를 추가할 플래그:  
   -D'TI\_AFE77XXD\_FUNC\_NAME\_PREFIX(funcName)=(ti\_afe77D\_##funcName)'
4. 채널 리매핑 기능을 원하지 않는 경우 컴파일 전에 -DENABLE\_RX\_CH\_REMAP, -DENABLE\_TX\_CH\_REMAP 및 -DENABLE\_FB\_CH\_REMAP 플래그 없이 컴파일합니다. 이렇게 하면 최적화를 위해 불필요한 로직이 컴파일되는 것을 방지할 수 있습니다. makefile에 -DUSE\_AFE77D\_LIB\_FORMAT\_2 플래그를 설정하지 마세요.
5. 채널 리매핑 기능을 사용하려면 afe77DInstDeviceInfo에서 rxChannelRemap, txChannelRemap, fbChannelRemap을 적절히 설정하세요.

# Version History

## Version 1.4

**tiAfe77D\_agc.c / tiAfe77D\_agc.h**:

1. ALC와 JESD 간의 지연 시간을 조정하는 **configureJesdAlcLatency** API가 추가되었습니다.

**tiAfe77D\_baseFunc.h**:

1. 베어 메탈을 사용하여 구동을 지원하기 위해 **hostMemRead**를 추가했습니다.

**tiAfe77D\_calibrations.c / tiAfe77D\_calibrations.h (1)**:

1. **getTxIQMCDelayCharFailcodes**, **advancedTxIqmcDelayChar**, **advancedExternalLoopBackDelayChar** 및 **getDpdLoopChannelResponse** 함수가 추가되었습니다.
2. 안정성을 개선하기 위해 **txIqmcDelayChar**와 **dpdDelayChar**를 업데이트했습니다.

**tiAfe77D\_controls.c / tiAfe77D\_controls.h**:

1. **configurePll\_FbNco**, **dynamicLoChange**, **readLo**, **dynamicLoChangeWrapper** 함수가 추가되었습니다.

**tiAfe77D\_dpdCfr.h**:

1. **cfrPeakCancellationEnableDisable**, **setCfrLowLatencyConfiguration**, **readCapturedDpdData**를 추가했습니다.
2. **getDpdPerfEnhanceFeature2ParamsEstimationStatus**, **readPcCfrWaveform**, **dpdModelSwitch**, **triggerDpdDataCaptureAtGivenTimeInstant**, **captureDpdDataInLog**의 버그가 수정되었습니다.

**tiAfe77D\_init.c**:

1. **afeDeviceBringup**의 버그를 수정해 일부 경우 Poll Serdes 상태에서 오류가 발생하지 않도록 했습니다.

**tiAfe77D\_init.h**:

1. **postBringUpSetting**을 추가했습니다.

**tiAfe77D\_rx.h**:

1. **rxIqmcReset**이 추가되었습니다.

**tiAfe77D\_serDes.c**:

1. 일부 경우 Poll Serdes 상태에서 오류가 발생하지 않도록 **pollSerdesLinkStatusAllLanes**의 버그를 수정했습니다.

**tiAfe77D\_tx.c**:

1. **txIqmcLolFreeze**가 업데이트되었습니다.

**tiAfe77D\_tx.h**:

1. **getTxIqmcLolLoopTrackedDelay** 함수를 추가했습니다.

## Version 1.3

**tiAfe77D\_agc.c**:

1. **readSmallStepAttackConfig**, **readSmallStepDecayConfig**, **readBigStepDecayConfig**, **readRxPowerAttackConfig** 함수의 버그가 수정되었습니다. 이전에는 set과 get 사이에 1 편차가 있었습니다.

**tiAfe77D\_calibrations.c**:

1. RX가 연결된 채널을 호스트에서 전환할 수 있는 지원이 추가되었습니다.

**tiAfe77D\_dpdCfr.c**:

1. 다중 캐리어의 경우 **pcCfrAutoWaveformConfig**의 버그 수정.
2. CFR 지연 시간을 읽기 위한 **readCfrLatency** 함수 추가.
3. **triggerDpdDataCaptureAtGivenTimeInstant**, **captureDpdDataInLog** 및 **dumpDpdDataInLog** 함수 추가.
4. **readPcCfrWaveform** 함수의 버그 수정.

**tiAfe77D\_dpdCfr.h**:

1. CFR 대기 시간을 읽기 위한 **readCfrLatency** 함수가 추가되었습니다.
2. **triggerDpdDataCaptureAtGivenTimeInstant**, **captureDpdDataInLog**, **dumpDpdDataInLog** 함수가 추가되었습니다.

**tiAfe77D\_fb.c / tiAfe77D\_fb.h**:

1. 기존 **fbDataMemCapture** 함수가 제거되었습니다.

## Version 1.2 – todo

## Version 1.1 – todo

## Version 0.4 - todo

# Data Structures

## Data Structures

### afe77D\_glbParamsUpdInfoDef

### afe77D\_macroParamsUpdInfoDef

### afe77DAgcSystemParamsStructDef

uint8\_t thresholdSa

* Small Step Attack의 임계값. 범위는 0dBFS~-63.75dBFS이며, -0.25dbfs 단계입니다. (-thresholdSa/4)는 프로그래밍된 dbFs 값입니다.

uint32\_t windowLenSa

* Small Step Attack의 윈도우 길이. 임계값을 넘는 샘플이 계산되는 윈도우 길이. N의 값에 대해 윈도우 길이는 Fs/8 클록의 2^N 사이클이 됩니다. ulWindowLen의 최대값은 23입니다.

uint8\_t stepSizeSa

* 작은 단계 공격을 위한 단계 크기. DSA 공격 시 DSA를 증가시키는 단계. 1LSB=1dB.

uint8\_t numHitsSa

* 스몰 스텝 공격에 대한 타격 횟수.

uint8\_t enableSa

* Small Step Attack을 활성화합니다. 0-비활성화. 1-활성화.

uint8\_t thresholdBa

* Big Step Attack의 임계값. 범위는 0dBFS~-63.75dBFS이며, -0.25dbfs 단계입니다. (-thresholdSa/4)는 프로그래밍된 dbFs 값이 됩니다.

uint32\_t windowLenBa

* Big Step Attack의 윈도우 길이. 임계값을 넘는 샘플이 계산되는 윈도우 길이. N의 값에 대해 윈도우 길이는 Fs/8 클록의 2^N 사이클이 됩니다. ulWindowLen의 최대값은 23입니다.

uint8\_t stepSizeBa

* 큰 스텝 공격을 위한 스텝 크기. DSA 공격 시 DSA를 증가시키는 스텝. 1LSB=1dB.

uint8\_t numHitsBa

* 빅 스텝 공격에 대한 히트 수.

uint8\_t enableBa

* 작은 큰 공격을 활성화합니다. 0-비활성화. 1-활성화.

uint8\_t thresholdSd

* Small Step Decay의 임계값. 범위는 0dBFS~-63.75dBFS이며, -0.25dbfs 단계입니다. (-thresholdSa/4)는 프로그래밍된 dbFs 값이 됩니다.

uint32\_t windowLenSd

* Small Step Decay의 윈도우 길이. 임계값을 넘는 샘플이 계산되는 윈도우 길이. N의 값에 대해 윈도우 길이는 Fs/8 클록의 2^N 사이클이 됩니다. ulWindowLen의 최대값은 23입니다.

uint8\_t numHitsSd

* 작은 단계 감소를 위한 단계 크기. DSA 공격 시 DSA를 감소시키는 단계. 1LSB=1dB.

uint8\_t stepSizeSd

* 작은 단계 감소에 대한 히트 수.

uint8\_t enableSd

* Small Step Decay를 활성화합니다. 0-비활성화. 1-활성화.

uint8\_t thresholdBd

* Big Step Decay의 임계값. 범위는 0dBFS~-63.75dBFS이며, -0.25dbfs 단계입니다. (-thresholdSa/4)는 프로그래밍된 dbFs 값이 됩니다.

uint32\_t windowLenBd

* Big Step Decay의 윈도우 길이. 임계값을 넘는 샘플이 계산되는 윈도우 길이. N의 값에 대해 윈도우 길이는 Fs/8 클록의 2^N 사이클이 됩니다. ulWindowLen의 최대값은 23입니다.

uint8\_t stepSizeBd

* 큰 단계 감소를 위한 단계 크기. DSA 공격 시 DSA를 감소시키는 단계. 1LSB=1dB.

uint8\_t numHitsBd

* Big Step Decay의 조회수.

uint8\_t enableBd

* Small Big Decay를 활성화합니다. 0-비활성화. 1-활성화.

uint8\_t enableEl

* 외부 LNA 제어를 활성화합니다. 0-비활성화. 1-활성화.

uint8\_t lnabypassGain

* 외부 LNA의 이득.

uint8\_t gainMargin

* 이득 마진.

uint8\_t dgcEnable

* DGC를 활성화합니다.

uint8\_t dgcMode

* 0: 이득 분배 후 IEEE 부동 소수점 모드
* 1: 예약됨
* 2: Coarse\_Fine 이득 모드. 모든 출력 샘플에 표시된 거친 이득. I와 Q 모두 복제됨Coarse\_Fine 이득 모드. 모든 출력 샘플에 표시된 거친 이득. I와 Q 모두 복제됨
* 3: Coarse\_Fine 이득 모드. (I,Q) 샘플에 함께 분배된 거친 이득
* 4: Coarse\_Fine 이득 모드. ALC 핀을 통해 전송된 거친 이득
* 5: Coarse\_Fine 이득 모드. ALC 핀을 통해 입력으로 제공되는 거친 이득

uint8\_t coarseIndexBits

* 0:"0 bits. Include Everything",
* 1:"1 bit",
* 2:"2 bits",
* 3:"3 bits (Only for DGC Mode 4 or 5)",
* 4:"4 bits"

uint8\_t coarseStep

* 0:"0dB",
* 1:"1dB",
* 2:"2dB",
* 3:"3dB",
* 4:"4dB",
* 5:"Not Supported",
* 6:"6dB",
* 7:"Not Supported",
* 8:"8dB"

uint8\_t floatingPointMode

* 0- Exponent>0이면 MSB를 보내지 않음.
* 1- 항상 MSB를 보냄.

uint8\_t floatingPointFormat

* 0- 2-bit Exponent, 13-bit Mantissa, 1-Bit Exponent
* 1- 3-bit Exponent, 12-bit Mantissa, 1-Bit Exponent
* 2- 4-bit Exponent, 11-bit Mantissa, 1-Bit Exponent

uint8\_t enableIa

* 내부 AGC를 활성화합니다.

uint8\_t gainControl - 외부 AGC 모드에서 유효

* 0-Fast DSA
* 1-Pin AGC 8- Pin
* 2-Internal AGC
* 3-SPI
* 4-Fast DSA
* 5-Pin AGC 4-Pin

uint16\_t extLnaBlankingTime

* 외부 LNA 블랭킹 시간.

uint8\_t thresholdPa

* 파워 어택의 임계값. 범위는 0dBFS~-63.75dBFS이며 -0.25dbfs 단계입니다. (-thresholdSa/4)는 프로그래밍된 dbFs 값이 됩니다.

uint32\_t windowLenPa

* Power Attack의 윈도우 길이. 임계값을 넘는 샘플이 계산되는 윈도우 길이. N의 값에 대해 윈도우 길이는 Fs/8 클록의 2^N 사이클이 됩니다. ulWindowLen의 최대값은 23입니다.

uint8\_t stepSizePa

* 파워 어택을 위한 스텝 크기. DSA 어택 시 DSA를 증가시키는 스텝. 1LSB=1dB.

uint8\_t enablePa

* 작은 큰 공격을 활성화합니다. 0-비활성화. 1-활성화.

uint8\_t thresholdPd

* 전력 감소에 대한 임계값. 범위는 -0.25dbfs 단위로 0dBFS~-63.75dBFS입니다. (-thresholdSa/4)는 프로그래밍된 dbFs 값이 됩니다.

uint32\_t windowLenPd

* Power Decay의 윈도우 길이. 임계값을 넘는 샘플이 계산되는 윈도우 길이. N의 값에 대해 윈도우 길이는 Fs/8 클록의 2^N 사이클이 됩니다. ulWindowLen의 최대값은 23입니다.

uint8\_t stepSizePd

* 파워 감소를 위한 히트 수. DSA 공격 시 DSA를 감소시키는 단계. 1LSB=1dB.

uint8\_t enablePd

* 전원 감쇠를 활성화합니다. 0-비활성화. 1-활성화.

### afe77DcfrCoreRawStatsDef

### afe77DcfrCoreStatsDef

### afe77DcfrPowerReadStatsDef

uint8\_t cfr\_stats\_validity

* 0- CFR Out Stats are invalid, 1- CFR Out Stats are valid.

int16\_t cfr\_max\_peak

* 이 값 0.01은 dBF 단위의 전력입니다.

int16\_t cfr\_avg\_pwr

* 이 값 0.01은 dBF 단위의 전력입니다.

### afe77DChainGainsDef

uint16\_t txAnaDigDsa

* TX DSA 설정. 1LSB = 0.125dB.

uint8\_t fbDsa

* FB DSA 설정. 1LSB = 0.5dB.

uint16\_t cfrOpGainVal

* 이득 값. 0.01dB 단계. 범위: 0~12dB.

uint16\_t cfrPeakLimit

* CFR 피크 제한. 1LSB = -0.1dB.

uint16\_t dpdCorrOutputGainVal

* 0.01dB 단위로 DPD 추정 이득을 표시합니다.

# Files

Bare Metal(베어메탈): 컴퓨팅 환경에서 소프트웨어가 \*\*운영체제(OS)\*\*나 중간 소프트웨어 없이 하드웨어에서 직접 실행되는 시스템을 의미합니다. 이 방식에서는 애플리케이션이 하드웨어 리소스(메모리, 프로세서, 입출력 장치 등)를 직접 제어합니다.