AFE80xx 软件编程应用手册

Lilian Chang

摘要

为了实现对 **AFE80xx** 的操作控制，我们需要对 **AFE** 底层寄存器进行大量的读写操作，这就增加了客户在使用及调试器件期间的复杂度。 **TI** 为了简化客户的软件开发流程，提供了基于 **C ++**的库和功能函数的底层源代码。 这些功能函数具有极高的设计灵活性，客户可以通过这些功能函数实现对 **AFE80xx** 芯片的动态控制。本文着重介绍了如何将 **TI** 提供的 **C** 函数库进行封装编译到主机软件中，以及 **API** 函数的使用方法。

目录

1 引言 2
2 CAPI 源码结构 3
2.1 源码结构层级 3
2.2 主要特点 4
2.3 实例化库 5
3 AFE80xx 初始化配置 7
3.1 初始化配置文档格式 7
3.1.1 原始日志格式(Raw log format) 7
3.1.2 十六进制格式（Hex format） 7
3.1.3 原始日志格式与十六进制格式对比 9
3.2 初始化函数 9
4 动态API 10
5 参考文献 12

图/表

Figure 1. AFE80xx 功能框图 2
Figure 2. Raw log format 示例 7
Figure 3. Hex format 示例 8
Figure 4. Operand 操作数定义 8
Figure 5. Opcode 查找表 8
Figure 6. API函数列表 10
Figure 7. API函数参数示例 11

Table 1. 原始日志形式与十六进制形式对比 9

1 引言

AFE80xx 是一款以射频采样架构为基础的高集成度收发器，AFE80xx 具有高性能，大带宽，多通道等特点。如下图所示 AFE 共集成了八个射频采样发射/接收通道和两个射频采样反馈通道。AFE80xx 的发射器和接收器链的动态范围高，可以生成和接收无线基站的 3G，4G 和 5G 信号，同时 AFE80xx 芯片大带宽的特点设备更适用于多频段 4G 和 5G 基站应用场景。AFE80xx 芯片内部集成了低噪声的 PLL, 内置 VCO 可以产生 12Ghz 的信号频率。接收 ADC 的最高采样频率为 4Gsps，可提供高达 400MHz 的有效信号带宽。每个接收链路包括一个 35dB（AFE8020 / AFE8022 为 25dB）范围的 DSA（数字步进衰减器），同时每个接收通道都有模拟峰值功率检测器以及数字峰值/功率检测器，用以协助外部或内部 AGC（自动增益控制器），以及可用于设备可靠性保护的过载检测器。DAC 最高采样率为 12Gsps，可支持 800MHz 的信号带宽。DAC 输出包括一个可变增益放大器（TXDSA），量程为 40 dB，模拟量为 1 dB，数字步长为 0.125 dB。反馈 ADC 最高采样率为 4GSPS，反馈链路上包含了 35dB（AFE8020 为 25dB）范围的 DSA，并且具有高达 800 MHz 的信号带宽。

为了简化客户的软件开发流程，TI 提供了基于 C ++的库和功能函数的底层源代码。 这些 C++的库被作为 API/功能函数去动态控制 AFE80xx 芯片，这些 API 并不处理初始化序列，而是处理设备在初始化完成后的动态控制功能。同时我们加入了很多设计灵活性以确保客户可以更加容易的将这些库文件集成在主机软件中（host library），并且无需额外的精力即可开发软件驱动程序。

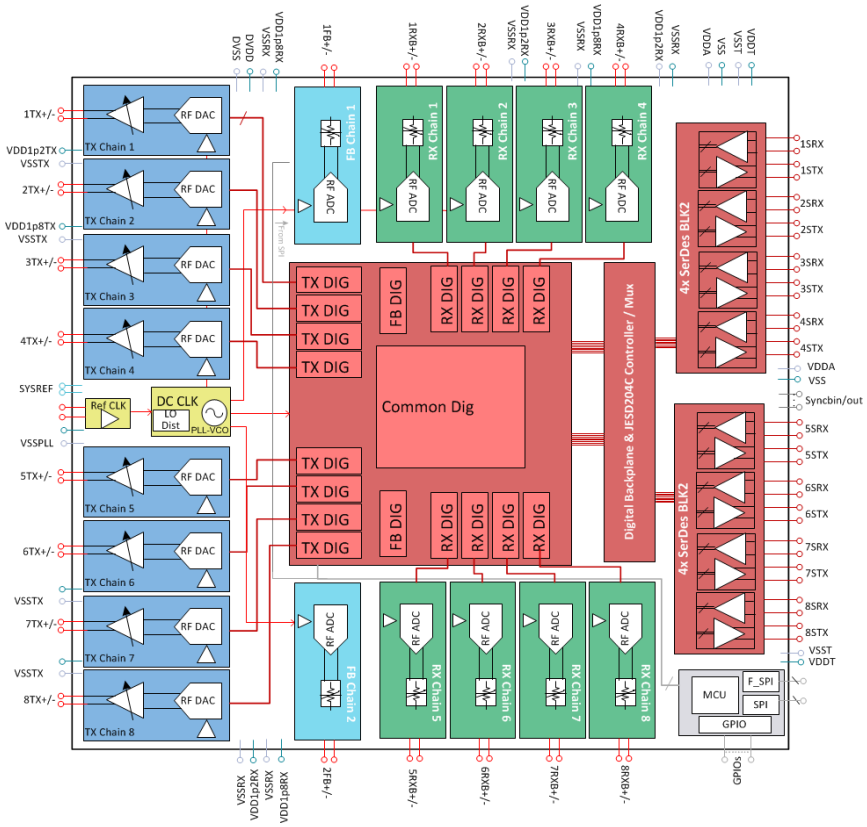
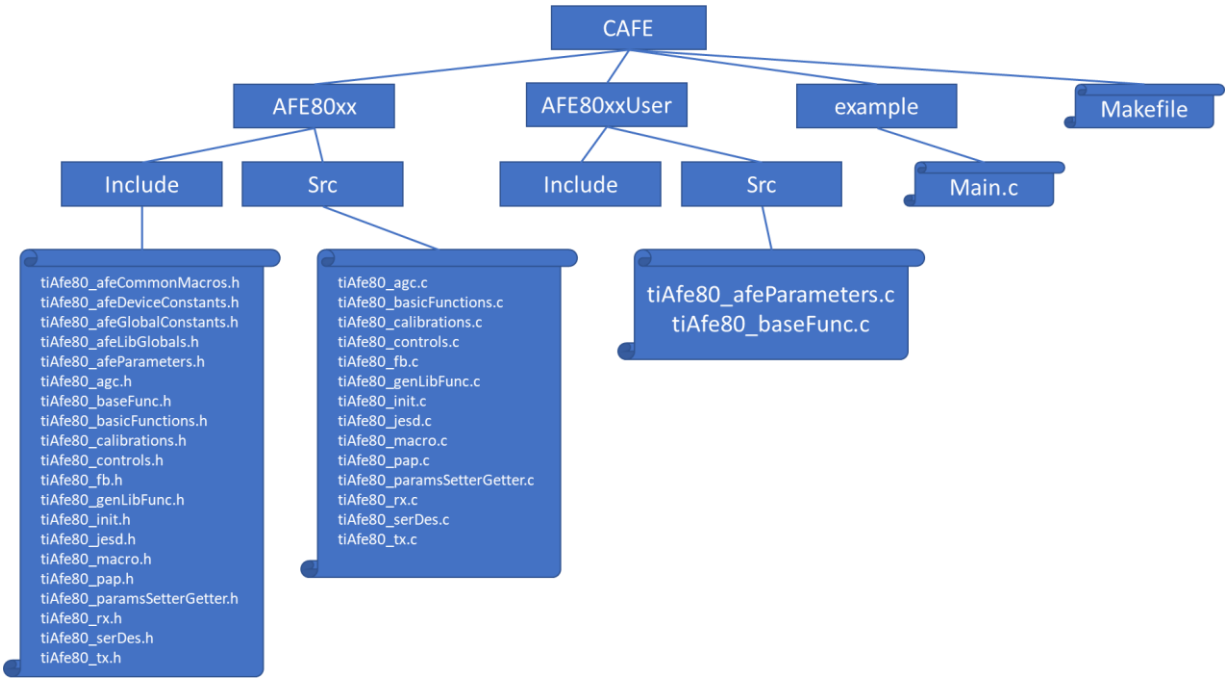


Figure 1. AFE80xx 功能框图

2 CAPI 源码结构

2.1 源码结构层级

AFE80xx C 函数源代码包含三个文件夹和一个独立的 makefile 文件，层级结构如下图所示：



1. AFE80xx: 该文件夹包含了所有 TI 提供的 API，用户无需修改直接编译即可使用。

2. Afe80xxUser: 此文件夹包含主控定义的主控驱动程序，需要用户进行改写：

   a. src\afeParameters.c：此文件完成 afeInst 结构体数组的初始化，该结构体数组的每个元素保存了对应 AFE 的初始化脚本中的配置参数和用户自定义通道重映射等信息。init_tiAfe80DeviceInfo_t() 函数根据 tiAfe80SysParams 结构体和用户通道重映射信息对的 afeInst 数组的每个元素进行初始化，该函数被 main 函数调用。tiAfe80SysParams 结构体包含了 AFE 初始化脚本中的参数（例如采样率，接口速率等），采用部分面向对象的思想，将 AFE 共性的参数打包到一个结构体中，防止过多的参数冗余传递到相关函数，结构体中的某些变量，还可以作为状态变量以记录当前 AFE 的状态。文件中提供的代码是所有 AFE 使用相同配置参数进行批量实例化的例子，同时我们的 API 也支持为每一片 AFE 配置不同的参数，以支持多模多频段的需求，代码也需要进行相对应的改写。

   b. src\baseFunc.c：包含和主控芯片驱动相关的函数，如 SPI 读写，日志记录及提供 sysref 等功能的函数，需要客户基于具体使用的不同主控平台重写这些函数以实现对应的功能。

3. example: 该文件夹包含一个 mian.c 文件，仅作为示例，用户可以参考该文件编写自己的 main 函数或在 main 函数中插入 AFE 相对应的代码片段。

4. makefile ：该文件是 TI 提供的 makefile 示例，需要用户根据实际编译环境和需求进行改写，修改可以包括但不仅限于：

a. 使用不同的编译器：gcc 编译器（CC = gcc），支持 C99 规范（CSTDFLAG = -std=c99）

b. 编译成可执行程序（Windows 下.exe 文件）；动态链接库（Windows 下为.dll 文件/Linux 下为.so 文件）或静态链接库（Lib 文件）

c. 添加用户文件路径，指定输出文件路径和文件名

d. 向源代码传入宏定义: 自定义函数名前缀

e. 自定义 make 和 clean 流程

2.2 主要特点

相比于上一代 AFE79xx 的 API，AAFE80xx API 有如下几个特点，

1. 完善了错误检查和处理功能: 每个函数执行完后都会有返回值指示该函数是否执行成功，返回值类型如下：

-TI_AFE_RET_EXEC_PASS：函数执行成功。

-TI_AFE_RET_EXEC_FAIL：函数执行失败，建议检查输入参数是否正确。

2. 支持一个主控操作多个 AFE 器件：API 中数据结构支持由同一主机控制多个 AFE 器件。由 afeInst 参数传递给每个具体的函数。共有通过两种方式可以完成传递此信息，实例化部分将对此进行进一步的说明。

3. 添加通道重映射功能：在客户系统实际使用场景中，我们经常遇到需要通道重新映射的情况，即系统的通道号与 AFE 实际使用的通道号并不相同。为了处理这种情况，AFE80xx 软件给客户提供了通过 API 去重新映射通道编号的功能。

4. 支持仿真模式：为了在没有硬件的情况下依然可以对软件进行测试，API 支持仿真模式。客户可以在 baseFunc.c 中设置 AFE80_LIBS_RUN_MODE 宏启用该模式，在此模式下，为了确保函数的正常执行，该程序将不执行任何读写或者轮询检查 SPI 的操作。

5. 函数重命名：AFE80xx 软件支持为所有的函数和 systemParams 的名称加上前缀，这可以通过在 MakeFile 中加入 gcc -D 参数向代码传递 TI_AFE80xx_FUNC_NAME_PREFIX 的宏定义来完成，不建议直接修改源代码，因为该宏定义位于 AFE80xx 文件夹下的文件内，API 版本更写迭代会覆盖掉之前客户的修改。

2.3 实例化库

在开始实例化库之前，我们把 AFE 的配置看作是一个抽象的对象，提取出所有 AFE 配置的共性，比如采样率，接口速率等的配置参数，我们称之为系统参数，这些参数构成了 AFE 配置这个对象的数据成员。对应到 C 代码中，这个对象对应的是 afe80SystemParamsStruct 结构体类型，而我们可以根据使用配置的不同，实例化不同的 tiAfe80SysParams 结构体。进一步的，还有一些参数，如日志级别，芯片序号，通道重映射等参数，把这些数据成员组合起来就构成了 AFE 这个对象。对应到 C 代码中，是 afe80InstDeviceInfo 结构体类型，而我们可以根据实际使用芯片的数量，为每一个 AFE 实例化一个 tiAfe80DeviceInfo_t 结构体。这些结构体组合在一起就是 tiAfe80DeviceInfo_t[n]结构体数组(n 为 AFE 器件数量)。为了 AFE 函数正常调用结构体中的数据成员，有两种方式初始和实例化库，对应两种不同的参数传递方式：

方法 1：在这种方法中，应将 afe80InstDeviceInfo 的指针传递给所有函数。

Instantiation Example:

```
afe80InstDeviceInfo tiAfe80DeviceInfo_t[2];
void init_tiAfe80DeviceInfo_t()
{
   setNumAfe80(2);

   extern afe80InstDeviceInfo tiAfe80DeviceInfo_t[2];
   tiAfe80DeviceInfo_t = tiAfe80DeviceInfo_t_inst;
   for (uint8_t i = 0; i < 2; i++)
   {
      tiAfe80DeviceInfo_t[i].afeId = i;
      for (uint8_t j = 0; j < 8; j++)
      {
         tiAfe80DeviceInfo_t[i].rxChannelRemap[j] = j;
         tiAfe80DeviceInfo_t[i].txChannelRemap[j] = j;
      }
      tiAfe80DeviceInfo_t[i].fbChannelRemap[0] = 0;
      tiAfe80DeviceInfo_t[i].fbChannelRemap[1] = 1;
      tiAfe80DeviceInfo_t[i].logLevel = AFE_LOG_LEVEL_SPILOG;
      tiAfe80DeviceInfo_t[i].halConfig = NULL;
   }
}
```

方法 2：在这种方法中，将 afe80InstDeviceInfo 的实例化为 tiAfe80DeviceInfo_t 数组，应按照 tiAfe80_afeParameters 中所示，按照 AFE 将相应的 AFE 索引传递给所有函数。

Instantiation Example:

```
afe80InstDeviceInfo tiAfe80DeviceInfo_t_inst[2]; // Example when the host is co
ntrolling 2 AFEs. For different number of AFEs, the size of the structure need to
be changed.
afe80InstDeviceInfo *tiAfe80DeviceInfo_t;
void init_tiAfe80DeviceInfo_t()
{
   setNumAfe80(2);
   extern afe80InstDeviceInfo tiAfe80DeviceInfo_t_inst[2];
   extern afe80InstDeviceInfo *tiAfe80DeviceInfo_t;
   tiAfe80DeviceInfo_t = tiAfe80DeviceInfo_t_inst;
   for (uint8_t i = 0; i < 2; i++)
   {
      tiAfe80DeviceInfo_t[i].afeId = i;
      for (uint8_t j = 0; j < 8; j++)
      {
         tiAfe80DeviceInfo_t[i].rxChannelRemap[j] = j;
         tiAfe80DeviceInfo_t[i].txChannelRemap[j] = j;
      }
      tiAfe80DeviceInfo_t[i].fbChannelRemap[0] = 0;
      tiAfe80DeviceInfo_t[i].fbChannelRemap[1] = 1;
      tiAfe80DeviceInfo_t[i].logLevel = AFE_LOG_LEVEL_SPILOG;
      tiAfe80DeviceInfo_t[i].halConfig = NULL;
   }
}
```

实例化过程：

1. 首先在 NUM_OF_AFE80 中设置需要由主机控制的 AFE 数量。这仅在上述方法 2 中有效。
2. 如果需要为函数名添加自定义前缀，请在 MakeFile 中添加-D 参数传入宏定义。如果未在 MakeFile 中添加此标志，则“ ti_afe80_”将被作为所有函数名的前缀。

   示例 1：标记为不带任何前缀添加：

   -D'TI_AFE80xx_FUNC_NAME_PREFIX(funcName)=(funcName)'

   示例 2：标记为添加“ ti_afe80_”作为前缀：

   -D'TI_AFE80xx_FUNC_NAME_PREFIX(funcName)=(ti_afe80_## funcName)'
3. 如果不使用通道重映射功能，请在编译之前注释掉-DENABLE_RX_CH_REMAP，-DENABLE_TX_CH_REMAP 和-DENABLE_FB_CH_REMAP。这样可以防止不必要的逻辑编译以进行优化。
4. 如果要使用通道重映射功能，请在 afe80InstDeviceInfo 中相应地设置 rxChannelRemap，txChannelRemap 和 fbChannelRemap 参数。
5. 在 afe80InstDeviceInfo 结构体中定义数组 tiAfe80DeviceInfo_t，并将其初始化为相应的值。
6. 在 baseFunc.c 中编写驱动程序。

3 AFE80xx 初始化配置

3.1 初始化配置文档格式

我们需要对 AFE80xx 进行初始化配置以正确使能器件进入工作模式。

在 AFE80xx 器件的初始化配置文件中，我们将同时支持两种初始化配置文档形式, 即传统的原始日志格式（Raw Log format） 和十六进制格式（Hex format）。这两种文件对于 AFE80xx 底层寄存器的操作是完全一样的，只是展现形式不同。

3.1.1 原始日志格式(Raw log format)

原始日志格式的结构与 AFE79 系列产品相同，如下图示例，包含了 SPIWrite/ SPIRead/ SPIPoll 等操作指示，注释用“//”表示。

示例：SPIWrite 0170,01,07 即表示对地址为 0x170 的寄存器进行 SPI 写操作，写入值为 0x01, 有效位为 bit[0:7]。

```
//START: Requesting/releasing SPI Access to PLL Pages

SPIWrite 0015,40,0,7    //digtop=0x1;   Address(0x15[7:6])
SPIWrite 0170,01,0,7    //pll_reg_spi_req_a=0x1;        Address(0x170[7:0])
SPIWrite 0540,00,0,7    //Property_520h_0_0=0x0;        Address(0x540[7:0])

SPIPoll 0171,0,0,01
SPIRead 0171,0,0

//Read  pll_reg_spi_a_ack=0x1(Meaning: );;      Address(0x171[7:0])


//END: Requesting/releasing SPI Access to PLL Pages

SPIWrite 0015,00,0,7    //digtop=0x0;   Address(0x15[7:6])
```

Figure 2. Raw log format 示例

3.1.2 十六进制格式（Hex format）

Hex format 是 AFE80xx 器件初始化文档的新格式，如下图示例。初始化配置被编写在以 32 位整数为序列的十六进制文件中。每个 32 位 bit 字代表了操作码 Opcode 或者操作数 Operand，不同的操作码用以指示不同的 SPIWrite/Read/Poll 等操作，注释用 “#” 表示。同时，我们还定义了 SPI burst mode。操作码将出现在一个字的高 16 位（称为字 0）中，字 0 的低 16 位和随后的字（Word1-n）为操作码。操作码和操作数的详细定义请参考“AFE80xx SW Library”手册。

```
#START: Enabling access to SERDES

0x00000000;     #Write
0x7010001D;     #jesd_subchip=0x1;      Address(0x1d[4:4])
0x00000000;     #Write
0x7002012C;     #apb_clk_disable=0x0;   Address(0x12c[0:0])
0x00000000;     #Write
0x7000012C;     #apb_clk_dithered_mode_en=0x0;  Address(0x12c[1:1])
0x00000000;     #Write
0x70010130;     #apb_clk_from_MCU_clk_en=0x1;   Address(0x130[0:0])
0x00000000;     #Write
0x70120088;     #serdesab_apb_page_addr_index=0x2;      Address(0x88[1:0])
0x00000000;     #Write
0x70120089;     #serdescd_apb_page_addr_index=0x2;      Address(0x89[1:0])
0x00000000;     #Write
0x70120088;     #serdesab_apb_mode_16b=0x1;     Address(0x88[5:4])
```

Figure 3. Hex format 示例

示例：如上图所示，根据 Opcode 查找表（下图 Figure5）可知 0x00000000 即表示进行 SPI 写操作，再参考对应 Opcode=0 写操作时的操作数定义，word1 的 bit[32:0]为操作数长度，bit[15:0]为寄存器地址，bit[23:16]为写寄存器值，bit[31:28]和 bit[27:24]分别表示写操作的有效位。0x7010001D 即表示对寄存器地址 0x001D 写值 0x10, 有效位为 bit[0:7].

操作码 Opcode=0 时的操作数 bit 字定义：

Word 0[31:16]	Word 0[15:0]	Word 1[31:28]	Word 1[27:24]	Word 1[23:16]	Word 1[15:0]
OP Code	xx	MSB	LSB	Data	Address

Figure 4. Operand 操作数定义

操作码 Opcode 查找表如下图 Figure 5 所示：

OpCode	Functionality
0	Write
1	Read
2	Read Check
3	Poll
4	Delay
5	Start a new step
6	Macro Execute
7	Give Pin Sysref
8	PLL SPI access
9	Load DSA Packet
10	Reserved
11	Burst Write
0b10xxx…	System Param Update

Figure 5. Opcode 查找表

3.1.3 原始日志格式与十六进制格式对比

原始日志相格式是 AFE 系列产品一直沿用的格式，相较于新的 Hex format,其优点是通俗易懂，不需要查找操作码表即可直观理解操作，特别是在芯片的调试阶段，更方便工程师进行 debug 工作。而新的十六进制格式也有其特有的优势，如文档更小，加载时间更短等。两种格式的详细对比如下表所示，客户可以根据不同的使用场景灵活的选择更适合的初始化文档格式。

原始日志形式与十六进制形式对比		
No.	Raw log format	Hex format
1	文档大小会稍微大一些	文档大小更小
2	初始化时间较长	初始化时间更短，因为Hex format可以处理SPI burst mode
3	运行后仅能返回操作是否成功或者失败指示	运行后可以打印日志，大多数情况下可以记录失败原因
4	对于不同的配置文件，需要客户更新相应的tiAfe80_afeParameters.c文件并且重新编译。需要确保所有tiAfe80SysParams的值与配置文件相同，任何的差异将使API无法正常运行。	Hex format可以自动更新tiAfe80SysParams中的参数值。对于不同的项目，客户不需要更改 tiAfe80_afeParameters.c 文件并重新编译。 对静态库的管理需求更少也降低了出错的风险。

Table 1. 原始日志形式与十六进制形式对比

3.2 初始化函数

AFE80xx 规定了初始化流程，包含上电时序要求及 JESD204C 建链流程，在对 AFE80xx 器件进行初始化配置时，我们支持两种方式实现。

方式一，即沿用 AFE79xx 的初始化配置方法，由 TI 提供原始日志格式的初始化配置文档和初始化流程要求，客户需要按照自己的软件进行格式转换，再将完整的初始化流程编写为可执行程序，并且合入到客户的软件版本中。我们当前不推荐使用这种方式，原因是对客户软件编程及维护的工作量要求较高，如果初始化配置文档有任何变更，都需要客户重新转换文档合入版本。

第二个方式是当前 TI 推荐使用的。即客户直接使用 TI 提供的编写好的初始化函数（API 程序），初始化程序里包含了 JESD204C 建链流程和交互功能，如要求 ASIC 发送 sysref, 查询链路状态和 error report 等告警打印等功能。该方式极大简化了软件编写和维护的工作量，使用时只需将项目对应的初始化配置文件路径传递到函数中，然后运行 API 即可。

如本文 3.1 章节所述，TI 提供了原始日志格式和十六进制格式两种初始化文档，需要调用不同的函数进行。初始化函数存在 AFE80xx 文件夹下的 tiAfe80_init.c 中，如下图 Figure 6 所示。其中 afeDeviceBringup 函数是调用 Hex format 文档时使用的，configAfeFromFileFormat0 对应 Raw log format 文档使用。

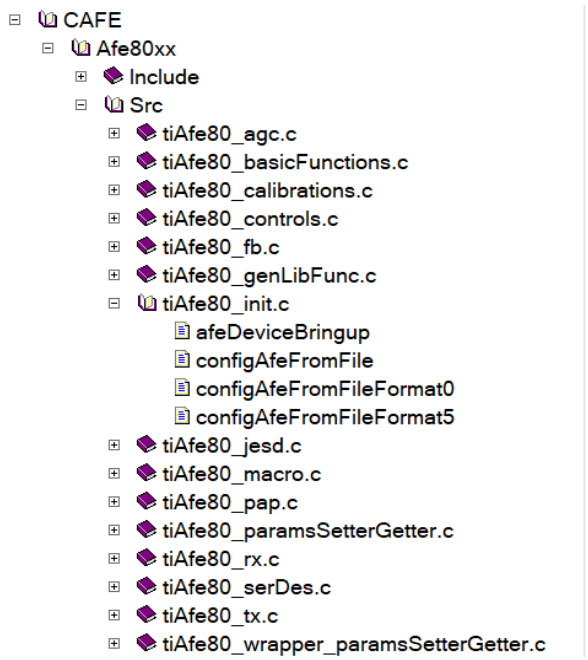


Figure 6. API 函数列表

4 动态 API

除了初始化函数外，还有一些功能函数不参与处理初始化序列，而是处理设备在初始化完成后的动态控制功能，我们将这类函数称为动态 API。所有的功能函数都存在 CAFE 的 AFE80xx 文件夹下，这部分的函数是由 TI 进行编写和维护，不需要客户编写，在使用时只需编译后调用。

动态 API 的功能基本可以分为以下三种类型：

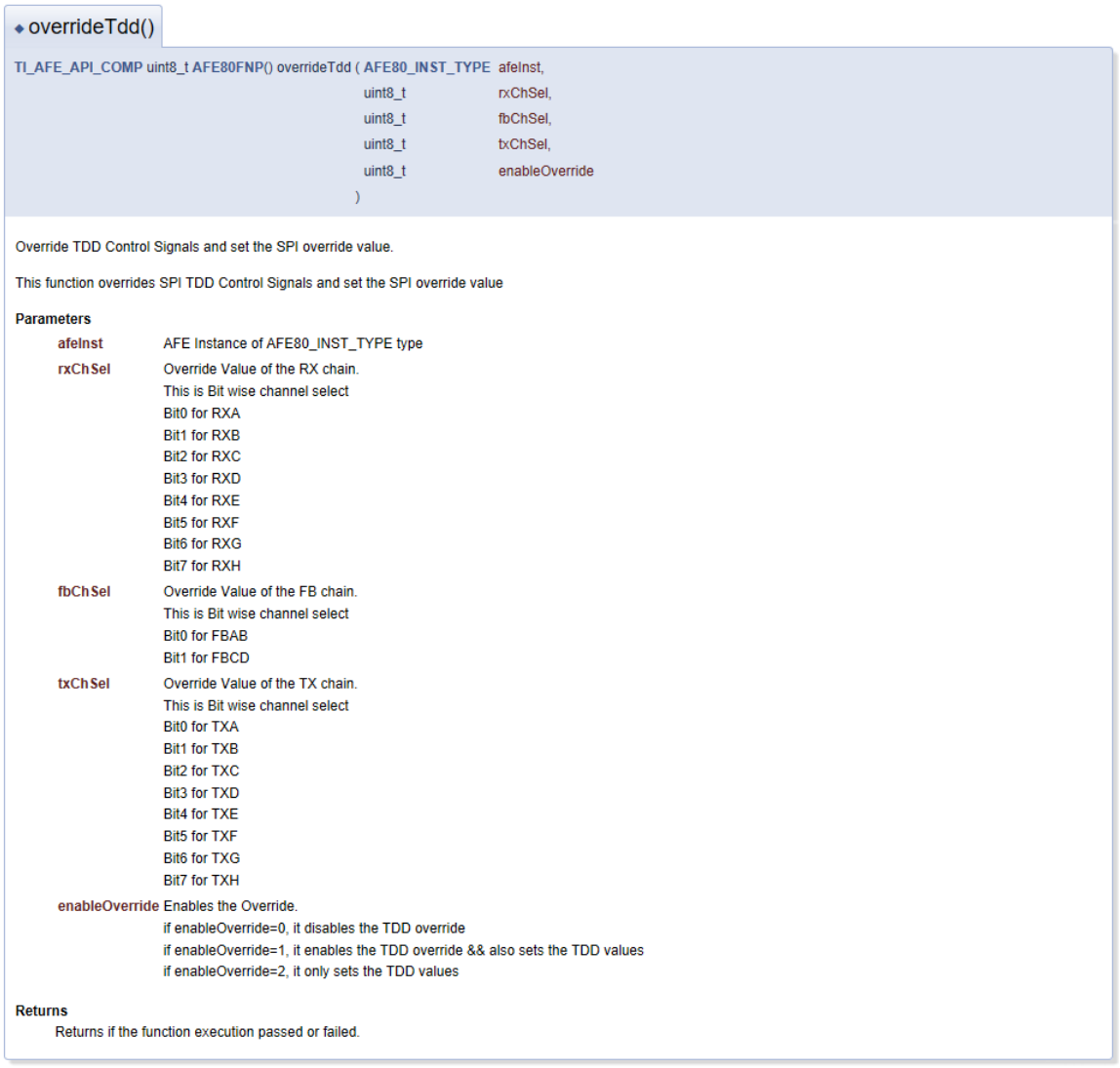
1. 基本的控制功能，如 tiAfe80_controls.c 中的 overrideTdd()，实现 GPIO pin 脚控制 TDD 切换功能， 和 tiAfe80_tx.c 中的 updateTxNcoDb()，实现 TX 通道的 NCO 频点更新等。

2. 状态检测功能，实现对器件状态的检测和上报，如 tiAfe80_jesd.c 中的 getJesdRxAlarms()等函数，可以查询当前 JESD 链路上是否有异常情况告警。

3. Debug 调试功能，比如在链路调试时经常使用的 JESD 发 test pattern 功能, 需要调用 tiAfe80_tx.c 中的 adcRampTestPattern()函数。

4. 参数配置功能，如 tiAfe80_agc.c 中的 API 可以实现客户实时修改 AGC(自动增益控制)功能中的配置参数。tiAfe80_pap.c 中的 API 可以实现客户实时修改 PAP(功放保护)功能中的相关配置参数。

下面介绍如何使用这些 API，打开 TI 提供的 afe80xxCLibsDocumentation 文件，在 CAFE 中的 AFE80xx 文件夹下可以按照类别和名称查找相应的 API，点击相应的 API 名字，右侧会出现对该 API 的功能介绍，参数和返回值说明介绍。如我们以 overrideTdd()为例，该功能实现的是对 TDD 控制 pin 脚信号的覆盖，即通过 SPI 对芯片 TX/RX/FB 通道的使能状态进行控制。根据实际配置需求，输入参数，afeInst 为芯片号，rxChSel/fbChSel/txChSel 分别为要控制的通道参数，每个参数的相应 bit 位置 1，表示该通道使能。enableOverride 为控制参数，选择是否要进行 SPI 覆盖写入或者关闭该功能，将控制权交回 pin 脚。



◆ overrideTdd()

```
TI_AFE_API_COMP uint8_t AFE80FNP() overrideTdd ( AFE80_INST_TYPE  afeInst,
                                                 uint8_t          rxChSel,
                                                 uint8_t          fbChSel,
                                                 uint8_t          txChSel,
                                                 uint8_t          enableOverride
                                               )
```

Override TDD Control Signals and set the SPI override value.

This function overrides SPI TDD Control Signals and set the SPI override value

Parameters

afeInst	AFE Instance of AFE80_INST_TYPE type
rxChSel	Override Value of the RX chain. This is Bit wise channel select Bit0 for RXA Bit1 for RXB Bit2 for RXC Bit3 for RXD Bit4 for RXE Bit5 for RXF Bit6 for RXG Bit7 for RXH
fbChSel	Override Value of the FB chain. This is Bit wise channel select Bit0 for FBAB Bit1 for FBCD
txChSel	Override Value of the TX chain. This is Bit wise channel select Bit0 for TXA Bit1 for TXB Bit2 for TXC Bit3 for TXD Bit4 for TXE Bit5 for TXF Bit6 for TXG Bit7 for TXH
enableOverride	Enables the Override. if enableOverride=0, it disables the TDD override if enableOverride=1, it enables the TDD override && also sets the TDD values if enableOverride=2, it only sets the TDD values

Returns

Returns if the function execution passed or failed.

Figure 7.　API 函数参数示例

每个动态 API 函数都将 AFE 实例标识符作为第一个参数，并且将其传递给所有子函数，包括用户在 AFE80xx User 文件下 tiAfe80_baseFunc.c 中实现的驱动程序函数。这可以用来确定需要对板上哪个 AFE 进行配置操作。根据上本文 2.3 章节提到的内容，我们有如下两种方式，如图 Figure 7 所示。

方式 1 的示例用法：

```
ti_afe80_overrideTdd(&tiAfe80DeviceInfo_t[afeId], 0x00, 0x3, 0xff, 1);
```

方式 2 的示例用法:

```
ti_afe80_overrideTdd(afeId, 0x00, 0x3, 0xff, 1);
```

5 参考文献

1. AFE80xx Datasheet
2. AFE80xx SW Library 7/23/2020
3. afe80xxCLibsDocumentation 10/28/2020

重要声明和免责声明

TI 提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保或其他要求。这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款 (https:www.ti.com.cn/zh-cn/legal/termsofsale.html) 或 ti.com.cn 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122