PXIE INTERFACE API底层函数说明文档

系统级驱动函数

sys_reset

函数作用：用于对指定子设备执行系统复位操作

函数输入：char* subid：子设备标识符

函数输出：返回值为 bool 类型，固定返回 true，表示复位操作已执行

执行过程：

1.第一次调用 **new_write_register** 函数，向地址为 0 的寄存器写入数据 1（触发复位开始）

2.第二次调用 **new_write_register**，向同一个地址 0 的寄存器写入数据 0（表示清除复位信号，结束复位过程）

sys_trigger

函数作用：整体系统触发函数，控制所有板子开始工作

函数输入：char* subid：子设备标识符

函数输出：返回 bool 类型，固定返回 true，表示触发信号序列已发送

执行过程：

1. 调用**new_write_register**向地址为4的寄存器写入0x01（触发激活）；
2. 再次调用 **new_write_register** 向同一地址4写入 0x00（恢复初始状态）；

说明：寄存器地址 4 是约定的触发控制寄存器

TTL板级驱动函数

sys_ttl_status

函数作用：配置TTL输入输出状态的函数

函数输入：char* subid：子设备标识符，用于指定要配置的具体子设备

uint32_t ttl_inout：TTL 输入输出状态配置值，用于设置 TTL 是工作在输入 模式还是输出模式

函数输出：返回值为 bool 类型，固定返回 true，表示 TTL 输入输出状态配置操作已发起

执行过程：调用 **new_write_register** 函数向硬件寄存器写入 TTL 配置信息

ttl_init

函数作用：初始化 TTL

函数输入：char* subid：子设备标识符

char* config_file：配置文件的路径或名称

函数输出：返回值为 bool 类型，其值由内部调用的 config_file_stream 函数返回。通常 true 表示 TTL 初始化成功

执行过程：以 config_file_stream 函数的返回值作为自身返回值

start_streaming

函数作用：用于启动 TTL 设备后台数据读取线程的驱动函数，支持手动开启数据流式读取，并可设置超时时间

函数输入：char* subid：子设备标识符；

DWORD timeout_ms：超时时间

函数输出：void

执行过程：

1. 调用find_device函数查找目标设备
2. 通过 SN_id 从 sys_device_paths 中获取对应设备的访问路径，创建 xdma_device 类的实例 ttl_dev_ptr（指向该设备的接口，用于后续数据读取）
3. 检查running标志，若已为true（线程已启动），则直接返回。否则将running设为true，并创建std::thread类型的reader_thread，传入线程函数reader_loop和超时时间timeout_ms，让线程在后台执行数据读取逻辑。

说明：reader_thread 是一个独立的后台线程，启动后会在后台持续运行。ttl_dev_ptr为指向 xdma_device 的指针。

get_latest_data

函数作用：从缓冲区中获取最新 TTL 设备数据

函数输入：char* out_buffer：输出缓冲区指针，用于存储读取到的数据

size_t max_len：最大读取长度，避免超出 out_buffer 的容量

函数输出：返回 size_t 类型，即实际读取到的数据长度

执行过程：

1. 创建std::lock_guard<std::mutex> 对象 lock，构造时自动锁定 buffer_mutex 互斥锁，确保当前函数访问缓冲区时，后台读数线程不会与写线程发生冲突。

2. 计算缓冲区中当前可用的未读取数据量（available）。
3. 取未读取数据量和最大允许读取长度中的较小值作为实际读取长度（to_read）
4. 读数据到out_buffer
5. 更新read_pos指针

stop_streaming

函数作用：停止通过start_streaming启动的后台读数线程，释放线程资源

函数输入：无

函数输出：Void

执行过程：

1. 首先判断 running 标志，IF running是false则return
2. 否则将 running设为 false，这个标志会被reader_loop函数检测到，促使线程结束数据读取操作
3. 调用 reader_thread.joinable() 检查线程是否可被等待，若可等待，则调用 reader_thread.join()，主线程会阻塞等待后台线程完全执行完毕

调用 delete ttl_dev_ptr 释放之前创建的 xdma_device设备接口实例，将 ttl_dev_ptr 设为 nullptr

AWG板级驱动函数

awg_ddr_init

函数作用：初始化 AWG设备中 DDR 存储器

函数输入：char* subid：子设备标识符

char* config_file：配置文件路径，该文件包含 DDR 初始化所需的参数

函数输出：返回值为 bool 类型，其值由内部调用的 config_file_stream 函数返回，通常 true 表示 DDR 初始化成功

执行过程：以 config_file_stream(config_file, subid, awg_card) 函数的返回值作为自身返回值，告知调用者 DDR 初始化操作的结果

awg_sync_delay_load

函数作用：配置 AWG设备同步延迟参数

函数输入：char* subid：子设备标识符；

uint32_t delay：同步延迟时间参数

函数输出：返回 bool 类型，固定返回 true

执行过程：

1. 调用new_write_register函数向地址为12的寄存器（同步延迟配置寄存器）写入delay值
2. 向地址16的寄存器进行激活延迟配置

awg_calib_status

函数作用：查询 AWG设备校准状态

函数输入：char* subid：子设备标识符

函数输出：返回 bool 类型

true：存在校准错误

false：校准正常

执行过程：

1. 调用new_read_register函数，读取地址为812的寄存器值，传入ven_id、dev_id、awg_card.create_sub_id(subid)及操作描述（awg_flash_status）
2. 该寄存器是硬件设计中约定的“校准状态寄存器”用来判断是否正确校准

awg_sync_delay_again

函数作用：对AWG设备的指定寄存器进行设置，重置同步延迟的计时逻辑，确保同步延迟功能按照之前配置的参数delay值重新开始工作

函数输入：char* subid：子设备标识符

函数输出：返回 bool 类型，固定返回 true

执行过程：

1. 第一次调用new_write_register向地址为8的寄存器写入1
2. 第二次调用new_write_register向同一地址8的寄存器写入0

awg_sync_delay_success

函数作用：用于通知板卡校准完成

函数输入：char* subid：子设备标识符

函数输出：返回 bool 类型，固定返回 true

执行过程：

1. 第一次调用new_write_register向地址为8的寄存器写入2
2. 第二次调用new_write_register向同一地址8的寄存器写入0

寄存器级驱动函数

new_write_register

函数作用：向指定设备的寄存器写入数据

函数输入：uint32_t write_data：要写入寄存器的数据值

uint32_t addr：寄存器地址，指定要写入数据的目标寄存器位置

std::string ven_id：厂商 ID

std::string dev_id：设备 ID

std::string sub_id：子设备 ID（用于标识设备下的子模块或子组件）

std::string fun_name：用于操作描述

函数输出： bool 类型，固定返回 true，表示寄存器写入操作已执行

执行过程：1.调用 find_device 函数，根据厂商ID、设备ID、子设备ID和操作描述（fun_name）查找设备，获取设备的序列号 SN_id

2.使用查找到的序列号 SN_id，从 sys_device_paths 数组中获取对应的设备路径，创建 xdma_device 类的实例 dev 以打开该设备。

3.通过 dev 调用 write_axil_reg 方法，将 write_data 写入到设备的 addr 地址所指定的寄存器中

底层功能级驱动函数

find_device

函数作用：用于在系统中查找匹配特定标识信息的设备，并返回硬件序列号（SN_id）

函数输入：std::string ven_id：厂商 ID

std::string dev_id：设备 ID

std::string sub_id：子设备 ID

std::string fun_name：调用该函数的功能名称，用于错误信息提示

函数输出：返回unsigned类型的设备序列号（SN_id）

执行过程：

1. 初始化SN_id和SN_found(找到标记)
2. 当flag_scan为0时，调用get_device_paths函数，通过GUID（PCIe总线上FPGA设备的接口GUID）获取所有相关设备路径。若未找到任何设备，抛出运行错误。若找到设备，将设备路径深拷贝到sys_device_paths数组，并把flag_scan设为1

3. 遍历 sys_device_paths 中的所有设备路径。对每个设备路径，检查是否同时包含 ven_id、dev_id 和 sub_id 这三个标识，找到第一个完全匹配的设备时，记录其索引 SN_id，递增 SN_found 并退出循环
4. 若未找到匹配设备，抛出错误。若找到匹配设备，返回其索引SN_id

get_device_paths

函数作用：基于 Windows 系统 API 的设备接口枚举函数，用于获取与指定 GUID相关的所有设备接口路径，返回这些路径的列表

函数输入：GUID guid设备接口的全局唯一标识符

函数输出：返回 std::vector<std::string> 类型的硬件设备接口路径

执行过程：

1. 调用 SetupDiGetClassDevs 函数，返回设备信息列表device_info
2. 初始化各种用于描述设备接口信息
3. 通过 for 循环，调用 SetupDiEnumDeviceInterfaces 函数，逐个枚举设备接口。每次返回与目标 GUID匹配的设备接口信息，存储到device_interface 结构体中。
4. 使用SetupDiGetDeviceInterfaceDetail获取设备接口的详细信息，函数参数中的device_interface 指定了要获取详细信息的设备接口
5. 设备路径的 device_paths 向量

该过程调用的Windows系统函数功能：

ØSetupDiGetClassDevs：创建并返回一个设备信息列表的句柄，该列表包含系统中与指定 GUID 匹配的设备接口信息

ØSetupDiEnumDeviceInterfaces：从 SetupDiGetClassDevs 返回的设备信息 列表中，枚举第 index 个设备接口的基础信息

ØSetupDiGetDeviceInterfaceDetail：将设备路径写入缓冲区

xdma_device类

功能介绍：主机与FPGA交互数据和配置的封装。

write_to_engine/read_from_engine通过 SGDMA 引擎实现主机与 FPGA 之间的数据传输；

read_axil_reg/write_axil_reg通过 AXI-Lite 寄存器读写实现对 FPGA 的配置和状态查询；