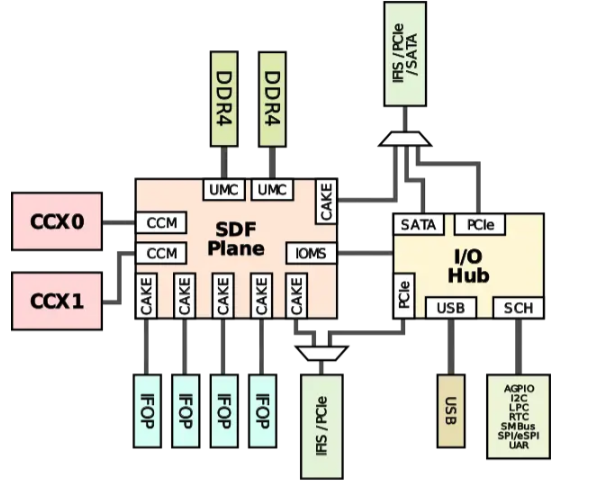
# 框架图



# AMD Scalable Data Fabric (SDF) 技术解析

​1. SDF 定义与定位​

​Scalable Data Fabric (SDF)​​ 是 AMD Infinity Fabric 架构的核心数据通信层，负责协调处理器内部所有组件（CPU 核心、内存控制器、I/O 设备等）之间的数据传输，并确保跨多芯片（Die）、多插槽（Socket）甚至多系统的一致性通信。

其本质是一种**全局数据高速公路**，支持灵活拓扑结构和扩展性。

2. 核心特性​

​跨层级一致性（Coherent Data Fabric）​​

SDF 通过**硬件级协议（如 MOESI）**维护所有缓存的一致性，确保多个 CPU 核心、内存控制器、I/O 设备访问数据时的正确性。

​示例​：当 CPU 核心 A 修改内存数据时，SDF 会自动通知其他核心和 I/O 设备更新缓存，无需软件干预。

​多维度扩展性​

​横向扩展​：支持单芯片内多核心集群（CCX）互联（如 Ryzen 的 CCD/CCD 结构）。

​纵向扩展​：通过 PCIe 链路跨多插槽（如 EPYC 的双路/四路服务器）或跨系统互联（理论支持）。

​拓扑灵活性​：支持总线、环形、网状等任意拓扑结构，避免传统架构的带宽瓶颈。

​统一时钟域设计​

SDF 平面与 DDR4 内存控制器（UMC）​共享 MEMCLK 频率​（例如 DDR4-2133 对应 SDF 频率 1066 MHz）。

​优势​：消除时钟域转换延迟，减少数据同步开销，提升实时性。

3. 模块功能详解​

基于图示的 ​Zeppelin SoC (Zen 架构)​​ 模块划分：

| **​模块​** | **​功能​** |
| --- | --- |
| ​**CCX (CPU Complex)​**​ | 包含 4 个 Zen 核心及共享 L3 缓存，是计算任务的最小物理单元。 |
| ​**CCM (Cache-Coherent Master)​**​ | 作为核心与 SDF 的接口，管理缓存一致性协议，调度核心间及核心与外设的数据传输。 |
| ​**SDF Plane**​ | 数据交换主干网，连接所有 CCM、IOMS 和 UMC，支持多路并行传输。 |
| ​**IOMS (I/O Master/Slave)​**​ | 处理 I/O Hub 的通信（如 PCIe、SATA、USB 控制器），负责外设与内存/核心的数据中转。 |
| ​**UMC (Unified Memory Controller)​**​ | 管理 DDR4 内存通道，直接接入 SDF，优化内存访问延迟与带宽。 |

4. 数据流示例​

**以 ​CPU 核心访问 PCIe 设备​ 为例：**

**​请求发起​：**

CCX 内的核心通过 CCM 向 SDF 发送 PCIe 设备数据请求。

**​路由决策​：**

SDF 检测目标地址是否属于本地 I/O Hub：

若属于，直接通过 IOMS 转发至 PCIe 控制器；

若属于远端插槽，通过 PCIe 链路跨插槽传输。

​数据返回​：

PCIe 设备响应数据经 IOMS 接收，由 SDF 路由回源 CCM，最终写入核心缓存或内存（UMC 管理）。

# ​CCM（Cache Coherence Manager）

**定义​：**

CCM 是 AMD 处理器中的「缓存一致性管家」，专管多核 CPU 之间数据同步问题，确保所有核心看到的共享数据都是最新版本，避免出现“你改了我不知道”的混乱局面。

**核心功能​**

**​数据一致性​：**

当 CPU 核心 A 修改了某个数据，CCM 立刻通知其他核心：“这数据已过期，快更新！”

​例子​：8 个核心同时处理一份 Excel 表格，CCM 确保所有核心看到的是实时修改后的内容。

**​缓存同步​：**

管理 L1/L2/L3 缓存的数据流动，减少核心间互相“抢数据”的冲突。

​例子​：核心 1 从内存读取数据到 L3 缓存，核心 2 想用同一数据时，CCM 直接分配副本，无需重复读取。

# CCX (CPU Complex)​​

**1. 定义**

**CCX（Core Complex） 是 AMD Zen 架构中的核心集群单元，代表处理器内部的最小物理核心组合模块。**

**每个 CCX 包含：**

**CPU 核心：4 个 Zen 架构物理核心（如 Zen 2、Zen 3）。**

**共享 L3 缓存：所有核心共享同一块 L3 缓存（Zen 2 为 16MB，Zen 3 为 32MB）。**

**一致性互连总线：核心间通过高速总线通信，降低延迟。**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2. 设计目的**

**模块化设计：将多个小规模核心集群（CCX）组合成多核处理器，提升良品率。**

**降低延迟：同一 CCX 内的核心共享缓存，数据交换无需跨芯片传输。**

**灵活扩展：通过 Infinity Fabric 总线连接多个 CCX，实现从 4 核到 64 核的平滑扩展。**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**3. 典型例子**

**(1) 消费级处理器：Ryzen 5 3600（Zen 2 架构）**

**CCX 配置：2 个 CCX（每个 CCX 含 3 个核心，总计 6 核 12 线程）。**

**L3 缓存：每个 CCX 共享 16MB，总 L3 缓存 32MB。**

**应用场景：游戏时，同一 CCX 内的核心优先调度，减少跨 CCX 通信延迟。**

**(2) 服务器级处理器：EPYC 7763（Zen 3 架构）**

**CCX 配置：8 个 CCD（每个 CCD 含 1 个 CCX，共 8 CCX × 8 核 = 64 核 128 线程）。**

**L3 缓存：每个 CCX 共享 32MB，总 L3 缓存 256MB。**

**优势：数据库处理时，同一 CCX 的 8 个核心可快速共享缓存数据，避免跨 CCD 访问。**

# ​Scalable Data Fabric SDF

SDF（可扩展数据架构）是 AMD ​Infinity Fabric 技术的**数据传输核心**，负责在芯片内部或跨芯片、跨服务器之间高效、低延迟地传输数据。

它是 AMD 处理器（如 Ryzen、EPYC）的“数据高速公路”，与 SCF（控制平面）协同工作，但分工明确：

​SCF​：负责发送**控制指令**（如“CPU 降频”“显卡加速”）。

​SDF​：负责实际**数据搬运**（如 CPU 计算数据、显卡渲染数据、内存读写数据）。

**场景 1：你玩《赛博朋克 2077》​​**

​CPU 计算物理效果​ → **数据**从 **CPU 核心（CCX）**通过 **SDF** 直送内存。

​显卡渲染 4K 画面​ → SDF 同时将内存中的贴图数据“灌”给显卡，全程不卡顿。

​SSD 加载新场景​ → SDF 为硬盘预留专用通道，数据直通内存，游戏无缝加载。

​结果​：所有数据流并行处理，帧率稳定 60 FPS，无卡顿。

**​场景 2：数据中心服务器处理海量请求​**

​用户请求 1​ → 由 CPU 芯片 A 处理，数据通过 SDF 跨芯片共享给芯片 B。

​用户请求 2​ → 芯片 B 直接访问芯片 A 的内存数据（无需复制），延迟降低 40%。

​多路服务器扩展​ → 4 颗 EPYC CPU 通过 SDF 互联，共享 1TB 内存，性能线性提升。

​结果​：云计算、AI 训练等任务效率暴增，成本反而更低。

# Scalable Control Fabric (SCF)

1. 定义与定位​

​Scalable Control Fabric (SCF)​​ 是 AMD Zen 架构及后续处理器中的**控制**平面基础设施，**负责处理器内部所有控制信号（非数据信号）的传输、同步与仲裁。**

**其核心功能包括：**

​系统管理​：温度/功耗监控、频率调节（通过 SMU，System Management Unit）。

​一致性控制​：缓存一致性协议（如 MOESI）的硬件级实现。

​多芯片协同​：跨多 Die、多 Socket 的指令分发与资源调度。

​2. 核心组件​

| **模块​** | **​功能​** |
| --- | --- |
| ​**SMU**​ | 系统管理单元，全局监控电压、温度、功耗，动态调节核心频率/电压（如 Precision Boost）。 |
| ​**IFIS SerDes**​ | 专用物理层链路（25Gbps 速率），用于跨 Die/Socket 的低延迟控制信号传输。 |
| ​**SCF 路由节点**​ | 分布式控制节点，解析指令目标地址并转发至目标模块（如 CPU 核心、内存控制器、PCIe 总线）。 |

3. 工作原理​

**​控制指令生成​：**

SMU 检测到温度超标 → 生成降频指令（CPPC: Core Performance Power Control）。

**​SCF 路由​：**

**指令通过 SCF 的专用通道（独立于数据总线）**广播至所有 CCX（CPU 核心集群）。

**​执行反馈​：**

目标模块（如 CPU 核心）执行指令后，通过 SCF 返回状态信号（如实际频率值）。

**​跨芯片扩展​：**

在多 Die/Socket 场景，IFIS SerDes 链路直接连接不同 SCF 域，确保全局状态同步（如 NUMA 架构下的内存访问策略）。

# Die（芯片裸片）

1. 定义​

​Die​ 是半导体制造中的核心单元，指从硅晶圆（Wafer）上切割下来的独立集成电路单元，包含完整的晶体管布局和功能电路。其本质是未经封装（未加引脚/散热结构）的裸芯片，是芯片物理设计的最终载体。

​**功能​：一个Die是具备完整电路功能的独立单元，**比如：

包含4个CPU核心 + 缓存 + 内存控制器

或包含GPU计算单元 + 显存接口

# **​UMC（统一内存控制器）**

1. 定义与核心功能​

​UMC（Unified Memory Controller）​​ 是 AMD Zen 架构中的集成化内存控制单元，**负责统一管理处理器对各类内存（DDR4/DDR5、HBM、显存）的访问，并协调多核心、多芯片间的内存资源调度。**

**其核心功能包括：**

​协议兼容​：支持 DDR4-3200 至 DDR5-6400、HBM2e 等内存标准。

​通道聚合​：将多物理通道（如 DDR5 双通道）虚拟为统一地址空间，提升有效带宽。

​NUMA 优化​：在 EPYC 多路服务器中维护非统一内存访问（NUMA）拓扑，降低跨节点延迟。

# UMCCH（Unified Memory Controller Channel Hub）

UMC 的通道控制枢纽，专门管理内存通道（Channel）的分配、调度与优化。

**​1. 层级结构​**

​UMC​ 是顶层模块，统筹所有内存相关操作。

​UMCCH​ 是 **UMC 的子模块**，专注于多通道内存的精细化控制。

**以 ​双通道 DDR5 内存写入​ 为例：**

​UMC 接收指令​：CPU 核心请求写入 128MB 数据。

​UMC 拆分任务​：将数据分为两部分（64MB + 64MB）。

​UMCCH 执行调度​：

通道 A（UMCCH 0）写入前 64MB，时序参数 tCL=40。

通道 B（UMCCH 1）写入后 64MB，时序参数 tCL=38（根据温度动态优化）。

​UMC 汇总状态​：确认两条通道写入完成，反馈给 CPU。

# NBIO（North Bridge I/O） 定义​：

NBIO（北桥输入输出）是 AMD 处理器中**集成的高性能 I/O 控制器**，**负责管理 CPU 与外部设备（如显卡、硬盘、USB 接口,PCIe接口等）之间的数据通信**。传统 PC 中，北桥是独立芯片（位于主板上），而 AMD 从 Zen 架构开始将北桥功能直接集成到 CPU 内部，大幅降低延迟、提升效率。

**NBIO 的三大核心任务​**

**​PCIe 通道分配​**

​干什么​：把 PCIe 通道（类似高速公路的车道）分给显卡、硬盘、网卡等设备。

​举个栗子🌰​：

显卡独占 PCIe 5.0 x16 车道（全速飙车）。

NVMe 固态硬盘分到 PCIe 4.0 x4 车道（独立通道，不和显卡抢路）。

万兆网卡再分一条 PCIe 3.0 x4 车道（各走各的，互不干扰）。

**​USB 接口调度​**

​干什么​：管理 USB 4、USB 3.2 等接口的数据流，确保外设（键盘、鼠标、U盘）实时响应。

​举个栗子🌰​：

你插着鼠标、键盘、外置 SSD 打游戏时：

​鼠标指令​ → 走 USB 2.0 通道（低延迟优先）。

​SSD 读盘​ → 走 USB 4 40Gbps 通道（大带宽优先）。

NBIO 像交警一样指挥，保证两者不堵车。

**​外设与内存/CPU 直连​**

​干什么​：让显卡、硬盘等设备直接访问内存，无需 CPU 中转（AMD 叫 ​SAM 技术）。

​举个栗子🌰​：

游戏加载时，显卡通过 NBIO 直接读取内存中的贴图数据，速度比传统架构（需 CPU 搬运）​快 15%​。

# FCH（Fusion Controller Hub）

定义​：

FCH 是 AMD 处理器中的南桥芯片​（South Bridge），负责管理主板上的**低速 I/O 设备​**（如 SATA 硬盘、USB 2.0、音频、网络等）。在早期 AMD 平台（如 APU 时代），FCH 与北桥（NBIO 的前身）分工合作，构成传统 PC 的「芯片组双桥架构」。

**​核心定位​：**

​北桥（NBIO）​​：管高速设备（PCIe、内存、显卡）。

​南桥（FCH）​​：管低速设备（硬盘、USB、声卡、网卡）。

# ​SMU（System Management Unit）

SMU 是 AMD 处理器中的「智能管家」，专门负责 **​监控和调节 CPU 的“身体状态”​**，**比如温度、电压、功耗、频率等**。它像一台 24 小时工作的健康监测仪，确保 CPU 不发烧（过热）、不饿着（电压不足）、不累垮（功耗超标），同时还能“超常发挥”（超频）。

**​温度管理​：**

​干什么​：实时检测 CPU 温度，触发降温策略（如降频、加速风扇）。

​举个栗子🌰​：你玩《原神》时 CPU 温度飙升到 90°C → SMU 立刻让 CPU 降频到 4.0 GHz，同时风扇转速拉到 3000 RPM。

**​电压与功耗调控​：**

​干什么​：动态调整 CPU 电压，既省电又防止“供电不足”。

​举个栗子🌰​：轻办公时，SMU 将电压从 1.4V 降到 1.1V，功耗减少 30%；游戏时再瞬间升压到 1.5V，性能拉满。

**​频率与性能优化​：**

​干什么​：根据负载自动超频（如 ​Precision Boost）或降频。

​举个栗子🌰​：

视频渲染时，SMU 让 CPU 全核冲到 5.2 GHz（超频模式）。

待机时，核心休眠，频率降到 0.8 GHz（省电模式）。

**​硬件安全保护​：**

​干什么​：防止电压/温度超标损坏硬件，甚至能检测恶意超频攻击。

​举个栗子🌰​：熊孩子强行用软件超频到 6.0 GHz → SMU 直接断电保护 CPU。

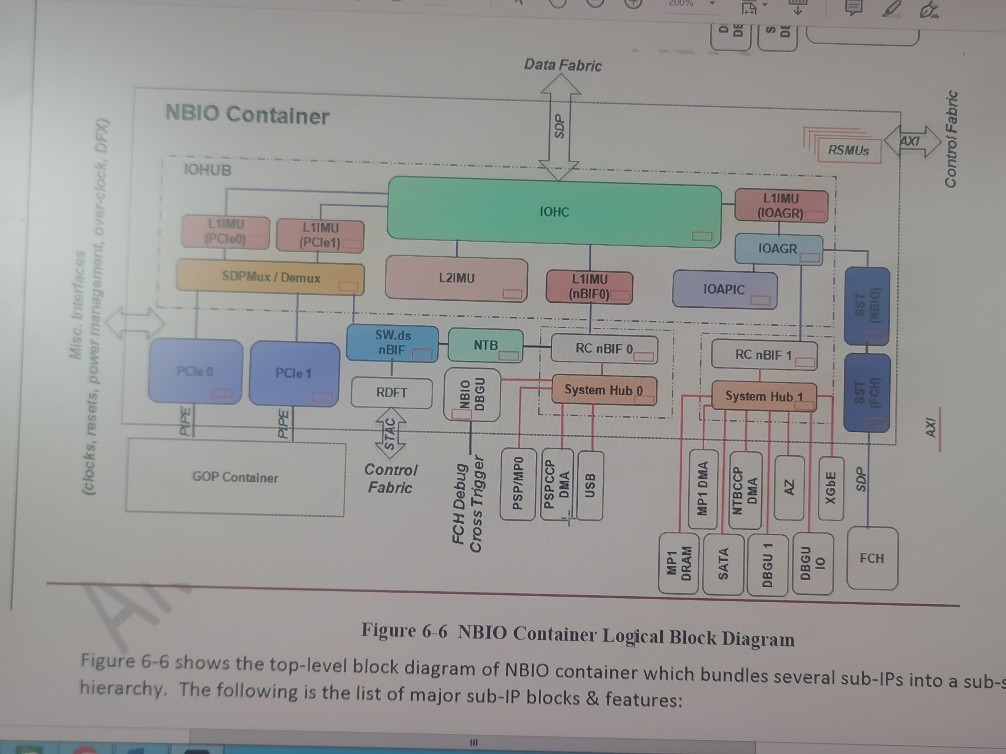
# ​MCM（Multi-Chip Module）

MCM 是 ​多芯片封装技术，简单说就是把多个小芯片（Chiplet）像拼乐高一样封装在一起，组成一个“大芯片”。

# AVFS（Adaptive Voltage and Frequency Scaling）

AVFS 是 AMD 处理器中的「智能节流阀」，能根据**任务需求自动调节电压和频率**，让 CPU 在“需要狂暴时全力输出，需要省电时躺平摸鱼”，榨干每一份性能的同时不浪费一滴电量。

# NBIO Container



NBIO Container​ 是 NBIO **内部**实现资源隔离与动态调度的模块，**核心任务包括：**

**​物理接口虚拟化​：**

将 PCIe x16 拆分为多个虚拟通道（如 4x x4），分配给不同设备。

​例子​：显卡用 x8，NVMe 硬盘用 x4，网卡用 x4，互不抢带宽。

**​优先级调度​：**

按任务需求分配 I/O 带宽（如游戏数据 > 下载任务）。

​例子​：游戏时显卡独占 90% PCIe 带宽，后台更新限速 10%。

**​硬件级隔离​：**

故障设备或恶意程序仅影响其所属 Container，不扩散到其他模块。

​例子​：外接 SSD 的 PCIe 链路故障 → 仅隔离对应 Container，显卡正常渲染。

**​虚拟化支持​：**

为虚拟机（VM）或容器分配独占 I/O 资源，避免“吵闹邻居”问题。

## RC nBIF（Reset Controller North Bridge Interface）

RC nBIF 是 AMD 处理器中的 ​**北桥接口复位控制器**，属于芯片内部电源与复位管理模块的一部分，**负责在系统启动、休眠唤醒、错误恢复等场景下，​安全重置北桥（NBIO）及相关高速接口（如 PCIe、USB 4）​**，确保硬件从异常状态快速恢复。

**核心功能​**

**​复位管理​：**

当系统检测到北桥或高速接口（如 PCIe 链路）的硬件错误时，RC nBIF 触发局部复位，避免全系统重启。

​例子​：显卡 PCIe 链路通信失败 → RC nBIF 仅复位 PCIe 控制器，而非重启整个 CPU。

**​电源状态切换​：**

在 CPU 休眠（C-State）或深度省电模式（如 S0ix）下，协调北桥接口的供电与时钟信号关闭/唤醒。

​例子​：笔记本合盖休眠 → RC nBIF 关闭 PCIe/USB 供电，开盖后 0.5 秒内唤醒。

**​错误隔离与恢复​：**

隔离故障模块（如某个 PCIe 端口），防止错误扩散到其他部件。

​例子​：外接 SSD 的 PCIe 链路电压异常 → RC nBIF 断开该端口，其他设备正常使用。

## ​IOHC（Input/Output Hub Controller）

IOHC 是 AMD 处理器中 ​I/O 控制的核心模块，属于 ​**IOHUB（Input/Output Hub）​​ 的一部分，**专门负责处理器与外部设备（如 PCIe 显卡、NVMe 硬盘、USB 外设等）之间的数据协议转换、流量调度和物理链路管理。

**核心功能​**

**​协议转换​：**

将 CPU 的原始数据“翻译”成外设能理解的协议（如 PCIe 数据包、USB 信号）。

​例子​：显卡的 PCIe 5.0 x16 信号 → IOHC 转换为 CPU 内部的 Infinity Fabric 协议。

**​带宽分配​：**

动态分配 PCIe、USB、SATA 等接口的带宽，避免设备争抢资源。

​例子​：同时使用显卡（PCIe 5.0 x16）和 NVMe 硬盘（PCIe 4.0 x4）时，IOHC 确保两者互不降速。

**​物理链路管理​：**

控制接口的电压、时钟、信号完整性（如 PCIe 链路训练和重试）。

​例子​：外接 SSD 因线材质量差导致信号错误 → IOHC 自动降低 PCIe 速率（从 5.0 降 4.0）保持稳定。

**​低功耗管理​：**

空闲时关闭未使用的接口供电，省电（如 USB 设备拔出后，IOHC 切断其电源）。

## L1IMU

L1IMU（Level 1 Interface Management Unit）是 AMD 处理器北桥接口（nBIF）中的 ​一级接口管理单元，属于 nBIF0（北桥接口模块的实例 0）的子模块，专门负责管理 CPU 与北桥（NBIO）之间的物理层通信，包括信号完整性、链路训练、错误检测等底层硬件控制。简单说，它是 CPU 与北桥“握手对话”的“硬件翻译官”。

**核心功能​**

**​物理层通信控制​：**

将 CPU 的电子信号转换为北桥能识别的物理波形（如 PCIe 差分信号）。

​例子​：显卡 PCIe 5.0 x16 的 32GT/s 高速信号，靠 L1IMU 保持信号稳定。

**​链路训练与协商​：**

设备连接时自动协商速率（如 PCIe 4.0 → 5.0）、通道宽度（x16 → x8 降级）。

​例子​：插入质量差的 PCIe 延长线 → L1IMU 自动降速到 PCIe 3.0 x8 确保稳定。

**​错误检测与恢复​：**

实时校验数据 CRC，发现错误触发重传或复位（部分功能与 RC nBIF 协作）。

​例子​：内存因电磁干扰传输出错 → L1IMU 丢弃坏数据包并请求重发。

**​功耗管理​：**

空闲时降低接口电压/时钟频率，省电（如 PCIe 链路进入 L1 低功耗状态）。

## ​SDP（Scalable Data Port）

SDP（可扩展数据端口）是 AMD 处理器中 **​IOHC（Input/Output Hub Controller）的专用数据通道**，负责在 IOHC 与其他模块（如内存控制器、跨芯片接口、加速器）之间传输数据。其核心作用是提供高带宽、低延迟、可扩展的互连通道，确保 I/O 数据流高效调度。

​SDP（Scalable Data Port）与 IOHC 的关系详解​

**​1. SDP 的核心功能​**

**​高速数据传输​：**

在 IOHC 与内存控制器（UMC）、PCIe 控制器、Infinity Fabric 之间搬运数据。

​带宽​：Zen 4 架构中单 SDP 通道可达 32GB/s（双向）。

​协议转换​：

将 IOHC 的 I/O 协议（如 PCIe TLP 包）转换为内存或跨芯片通信协议（如 Infinity Fabric 数据包）。

**​优先级仲裁​：**

按任务类型（如实时渲染、存储备份）动态分配带宽和优先级。

**​错误恢复​：**

检测传输错误并触发重传（部分功能与 L1IMU/RC nBIF 协作）。

## **nBIF（North Bridge Interface）**

nBIF 是 AMD 处理器中的 ​北桥接口模块，属于 ​NBIO（北桥 I/O）​​ 的核心组件，负责管理 **CPU 与北桥（North Bridge）之间的 ​物理层通信**，是NBIO的“物理层执行模块”。包括高速接口（如 PCIe、内存控制器、芯片间互联）的协议转换、链路训练和错误恢复。简单说，它是 CPU 与外部设备“握手对话”的 ​硬件翻译官 + 交警。

# [\\hygon.cn\doc\Private\SW\platform\BIOS\BIOS Image\Hygon\SL2\D3\043-132\043](file:///\\hygon.cn\doc\Private\SW\platform\BIOS\BIOS%20Image\Hygon\SL2\D3\043-132\043)

# **SMN（System Management Network）​**

​**定义**​：  
SMN（System Management Network，系统管理网络）是 AMD 芯片内部的一条**专用总线**，用于访问和管理硬件模块的**配置寄存器**。

​**作用**​：

集中管理 CPU、芯片组、PCIe、USB 等硬件模块的配置。

提供统一的接口访问硬件寄存器，无需通过内存映射 I/O（MMIO）。

​**类比**​：  
类似于汽车的“中央控制总线”，所有电子设备（如引擎、空调、音响）的配置都通过这条总线调整。

## **SMN地址**

**定义**​：  
SMN地址是 SMN 总线上用于**定位硬件寄存器**的逻辑地址。

​**特点**​：

​**独立于物理内存地址**​：SMN地址不与物理内存或虚拟内存重叠。

​**芯片级唯一性**​：每个硬件模块的寄存器在 SMN 地址空间内有固定位置。

​**示例**​：

​**USB PHY 配置寄存器**的 SMN 地址可能为 0xA580000。

​**PCIe 控制寄存器**的 SMN 地址可能为 0x1C00000。

# 特殊函数/宏

## FchSmnWrite

IohcBus:IO hc总线ID   
value：指向寄存器值的指针 SmnAddress:寄存器SMN地址   
VOID **FchSmnWrite**(UINT32 IohcBus, UINT32 SmnAddress, UINT32 \*value,HYGON\_CONFIG\_PARAMS \*StdHeader){ **SmnRegisterWrite**(IohcBus,SmnAddress,Value,0); }

## FchSmnRead

VOID FchSmnRead(UINT32 IohcBus, UINT32 SmnAddress, UINT32 \*value,HYGON\_CONFIG\_PARAMS \*StdHeader)  
{ **SmnRegisterRead**(IohcBus,SmnAddress,Value); }

## FchSmnRW

VOID **FchSmnRW**(UINT32 IohcBus, UINT32 SmnAddress,UINT32 AndMask,UINT32 OrMask,HYGON\_CONFIG\_PARAMS \*StdHeader) {   
UINT32 RegValue;   
**FchSmnRead**(IohcBus,SmnAddress,&RegValue,StdHeader);   
RegValue&=AndMask;

RegValue|=OrMask;

FchSmnWrite(IohcBus,SmnAddress,&RegValue,StdHeader); }

## USB\_SPACE\_HYGX

计算USB控制器的某个Address寄存器的实际起始地址。

当Address==0时，为USB控制器地址。

#define **USB\_SPACE\_HYGX**(PhysicalDiedId,UsbIndex,Address) (UINT32)(0x10000000+( PhysicalDiedId<<28)+(0x600000\*UsbIndex)+Address)

将物理设备 ID、USB 控制器索引和寄存器偏移组合成一个 32 位物理地址  
 #define **USB\_SPACE\_HYGX**(PhysicalDiedId,UsbIndex,Address) (UINT32)(0x10000000+( PhysicalDiedId<<28)+(0x600000\*UsbIndex)+Address)  
地址计算​ ​基地址​：0x10000000 ​物理模块偏移​：PhysicalDiedId << 28 → 0 << 28 = 0x0 PhysicalDiedId 左移 28 位，每个模块间隔 ​256MB​（0x10000000）。 ​USB 控制器偏移​：0x600000 \* UsbIndex → 0x600000 \* 1 = 0x600000 每个 USB 控制器间隔 ​6MB​（0x600000）。 ​寄存器偏移​：0x10 Address 精确定位到具体寄存器（如 0x10 为控制寄存器，0x14 为状态寄存器）

## USB\_REMU\_SPACE\_HYGX

#define **USB\_REMU\_SPACE\_HYGX**(UsbIndex,Address) (Address+(UsbIndex<<7))   
根据 USB 控制器的索引号（UsbIndex）和基础偏移地址（Address），计算最终的寄存器或内存地址 每个 USB 控制器或端口的寄存器组在内存中按 ​128 字节的间隔​ 排列.

## USB0PHY\_APPERTURE\_ID\_HYGX

//标识 ​USB0 PHY 的硬件配置空间（Aperture）的**索引号**   
#define USB0PHY\_APPERTURE\_ID\_HYGX 0x0A5

0x0A5<<20+0x80000 是0x0A580000

## USB0PHY\_APPERTURE\_ID\_HYMX

#define USB0PHY\_APPERTURE\_ID\_HYMX 0x16D

0x16D<<20是0x16D00000

0x16D00000+0x80000=0x16D80000

## FCH\_SMU\_USB\_CONTAINER\_HYGX

//**计算 ​USB0 PHY 配置寄存器的基地址​（Base Address）**。该地址用于访问 **USB PHY 的寄存器组**（如调整电压、时钟、阻抗等参数）。   
#define FCH\_SMU\_USB\_CONTAINER\_HYGX ((**USB0PHY\_APPERTURE\_ID\_HYGX**<<20)+0x80000) //0x0A580000ul

## FCH\_TS\_USB\_SMU\_ASSIST\_POWER\_STATE\_CTL\_HYGX

#define FCH\_TS\_USB\_SMU\_ASSIST\_POWER\_STATE\_CTL\_HYGX (**FCH\_SMU\_USB\_CONTAINER\_HYGX**+0x1C)

用于定义 ​**USB** 控制器**电源状态控制寄存器**​ 的地址。以下是详细解析：该寄存器用于配置 USB 控制器的 ​**电源管理模式**​（如低功耗状态切换、唤醒功能等）。