# Hygon BIOS Porting Guide

| | | | |
|---|---|---|---|
| 文件编号： | | QR-SW-003-001 | 版本：V1.3 |
| 文件发行日期： | | 2020 年 04 月 09 日 | |
| 文件状态：<br>☐ 草稿<br>■ 正式发布<br>☐ 正在修改 | 文档编号： | HYGON-SW-XXX-001 | |
| | 编　　写： | | |
| | 审　　核： | | |
| | 审　　批： | | |
| | 文档受控状态： | ■受控　　☐作废保留 | |

# 修订历史

| 修订前版本 | 修订内容 | 完成日期 | 修订人 | 修订后版本 |
|---|---|---|---|---|
| | 创建文档 | 2018/07/06 | | 1.0 |
| 1.0 | 修改文档 | 2019/12/20 | | 1.1 |
| 1.1 | 增加 5.6 节 Data Fabric IO 和 MMIO Resource 管理介绍 | 2020/03/06 | | 1.2 |
| 1.2 | 增加使能热插拔功能介绍 | 2020/04/09 | | 1.3 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

注：“草稿”状态的文档版本为 0.Y.Z，Y≥0，Z>0，Y、Z 的数值不断累加；

“正式发布”状态的文档版本为 X.Y，X≥1，Y≥0，且 X、Y 值不断累加；

“正在修改”状态的文档指对“正式发布”后的文档进行修改，文档版本为 X.Y.Z，其中 X.Y 同修改之前的文档版本号，Z>0，Z 的数值不断累加。

# 目录

# 1 Overview

## 1.1 Goal

本文档的目的是为了帮助用户更快地将 CRB BIOS 移植到用户的平台上。
本文档只覆盖 Memory、GPIO、PCIe、SATA、USB 等的配置。

## 1.2 Intended Audience

BIOS 开发者。

## 1.3 Acronyms

表格 1-1 缩略词

| 缩略词 | 全称 |
|---|---|
| APCB | AGESA Platform Control Block |
| | |

## 1.4 References

表格 1-2 参考文献

| 简称 | 描述 |
|---|---|
| | |

# 2 BIOS Flash Components

## 2.1 Layout

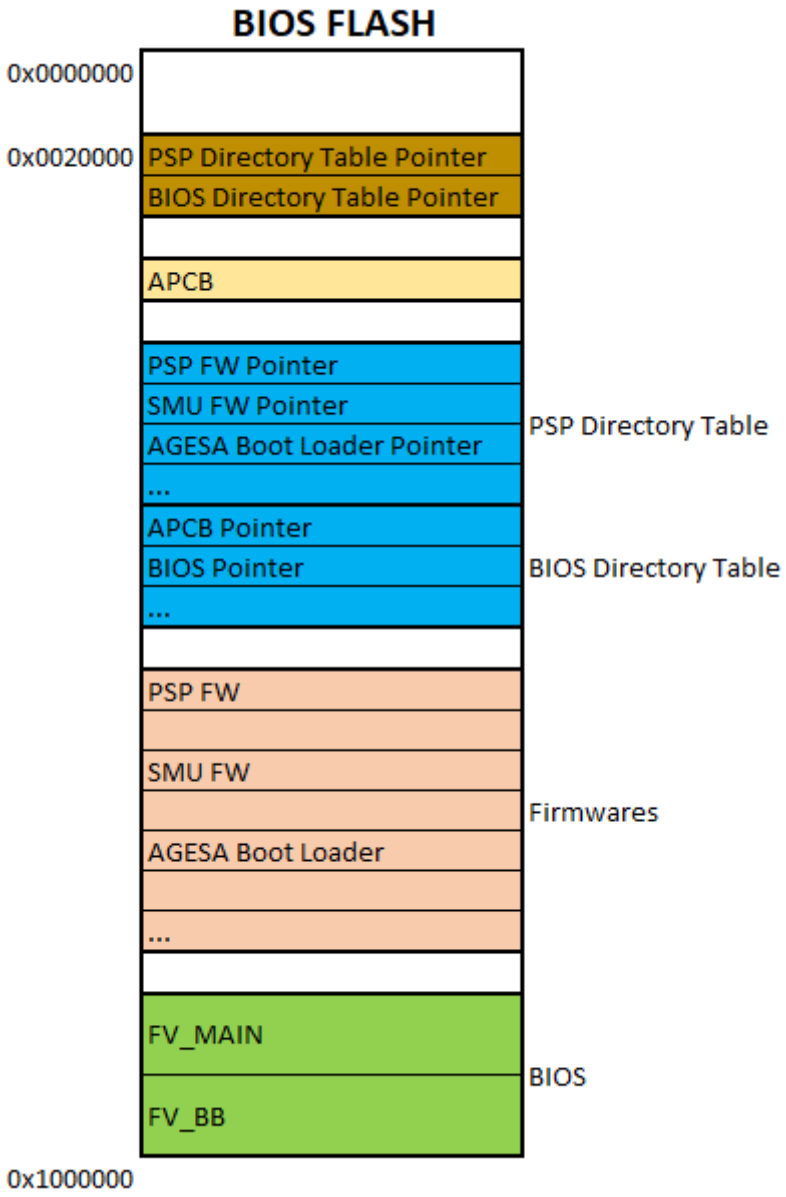



图 2-1 BIOS Flash Layout

## 2.2 Directory Table Bases

Directory Table Base 位于 Flash 偏移 0x20000 处的结构体中。PSP Directory Table Base 位于该结构体的偏移 0x20 处，BIOS Directory Table Base 位于偏移 0x24 处。这两个基地址并不是其目标在 Flash 中的偏移，而是将 BIOS Flash 的内容映射至内存地址区间 [4G-16MB, 4G) 之后其目标的地址。

## 2.3 PSP Directory Table

PSP Directory Table 包含了各 Firmware 在 Flash 中的位置信息。

## 2.4 BIOS Directory Table

BIOS Directory Table 包含了 APCB、PMU，以及 BIOS 的 FV_BB 在 Flash 中的位置信息。

## 2.5 Firmware

该文档提到的固件指的是 CPU/SoC 的固件，区别于 BMC/CPLD 的固件。

CPU/SoC 的固件包括：PSP Boot Loader，SMU，AGESA Boot Loader 等。

## 2.6 BIOS

客户从 IBV 处获得的 UEFI BIOS 代码包含了 AGESA PI。AGESA PI 是 Hygon 提供的与 Hygon CPU/SoC 相关的 SEC、PEI、DXE、SMM 等模块的集合。

## 2.7 APCB

APCB 是将平台配置信息打包而成的二进制数据块，在 POST 阶段，AGESA Boot Loader 会读取 APCB，然后根据配置对平台进行初始化。

# 3 BIOS/Firmware Execute Sequence

BIOS 和固件的执行顺序和交互关系如图 3-1 所示。图中只包含与用户移植代码过程中需要关心的内容。

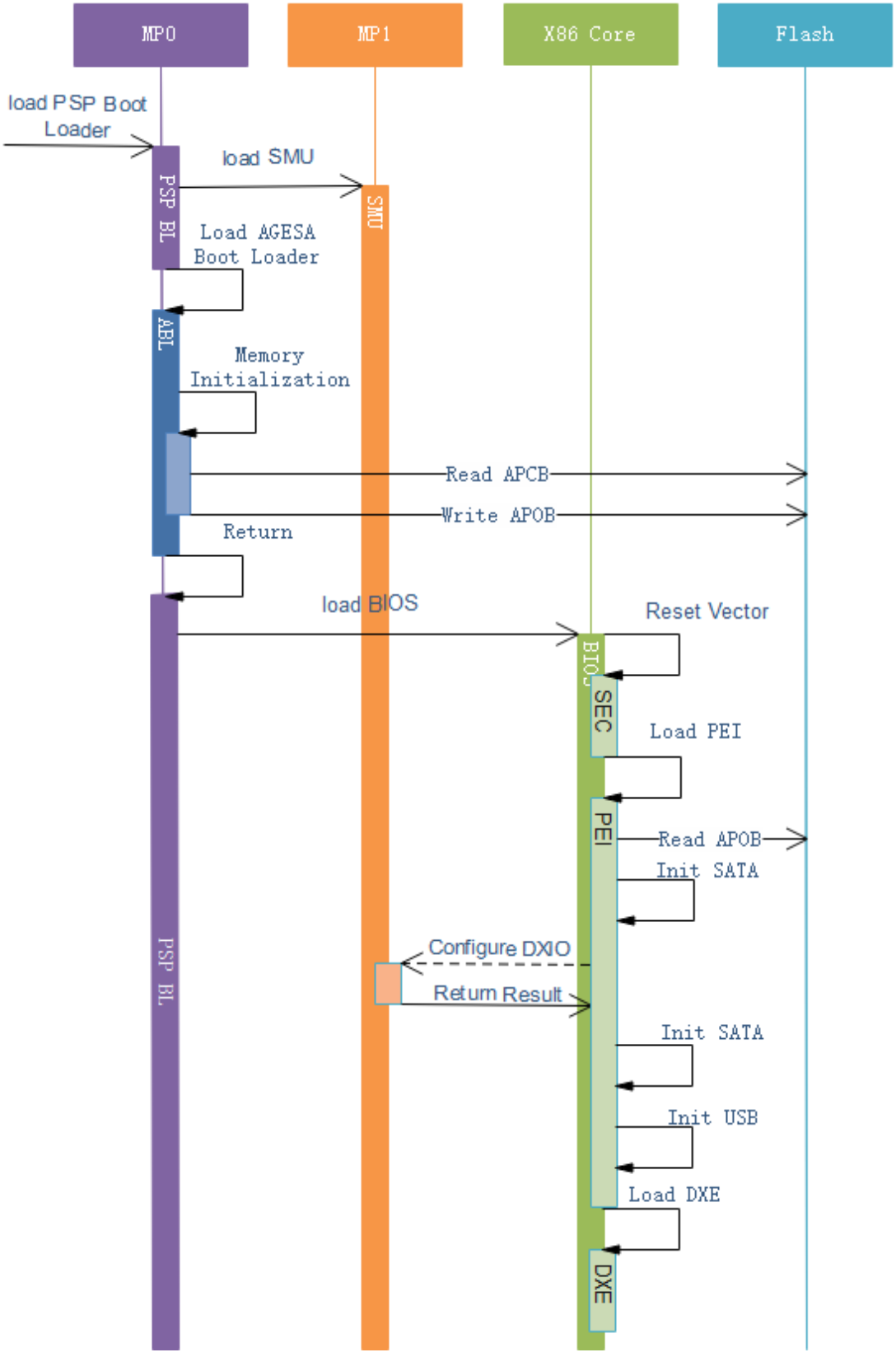



图 3-1 BIOS/Firmware Execute Sequence

# 4 AGESA Code Hierarchy

AGESA 提供了 4 个 Package，内容列于表格 4-1 中。

表格 4-1 AGESA Packages

| 目录 | 内容 |
|---|---|
| **AgesaModulePkg** | 这个 Package 包含了 AGESA 的内部接口和构建配置文件。用户不需要修改这个 Package 中的文件。 |
| **AgesaPkg** | 这个 Package 包含了所有的外部接口和构建配置文件。 |
| **AmdCbsPkg** | 这个 Package 包含了创建 BIOS Setup 页面的代码。 |
| **AmdCpmPkg** | 这个 Package 包含了平台相关的配置代码。 |
| **HygonWrapperPkg** | 这个 Package 包含了平台的环境初始化配置代码。 |

在 BIOS boot flow 里，建议按照以下顺序进行添加表格 4-1 中的各 package 相关 fdf 文件：

PEI 部分：HygonWrapperPkg.inc.fdf -> AgesaSp2ZpModulePkg.pei.inc.fdf -> CbsZeppelin.pei.inc.fdf ->CpmXXX.pei.inc.fdf

DXE 部分：AgesaSp3ZpModulePkg.dxe.inc.fdf -> CbsZeppelin.dxe.inc.fdf

# 5 Porting

要适配一个新的主板，可以根据新主板的 Socket 数量和 DIMM 插槽布局选择一个相似的 CRB 主板代码作为基础，根据主板的具体设计，对 Memory，GPIO，PCIe，SATA，USB 等相关配置进行修改来完成。

Hygon CRB BIOS 与主板适配相关的代码和数据结构位于 AgesaPkg/Addendum/Apcb 和 AmdCpmPkg/Addendum/Oem 这两个目录下，某些配置需要通过 PCD 或者搭配 PCD 完成。

## 5.1 CRB

表格 5-1 Hygon CRB 配置

| CRB | Package 类型 | Socket 数量 | Channels/Socket | DIMMs/Channel |
|---|---|---|---|---|
| **Hygon35N16** | SL1 | 1 | 8 | 2 |
| **Hygon65N32** | SL1 | 2 | 8 | 2 |
| **HygonDM1SLT** | DM1 | 1 | 2 | 2 |
| **Hygon52D16** | SL1r2 | 2 | 4 | 2 |
| **Hygon35N16** | SL1r2 | 1 | 4 | 2 |
| **Hygon65N32** | SL1r2 | 2 | 4 | 2 |

## 5.2 Memory

Memory 的配置包括 DIMM SPD I2C 地址配置和插槽丝印配置，位于 APCB 代码。

### 5.2.1 SPD

本节描述了 AGESA Boot Loader 发现 DIMM 布局所需的信息。根据主板设定，在文件 AgesaPkg/Addendum/Apcb/<Platform>/Include/ApcbCustomizedDefinitions.h 中修改如下信息。

表格 5-2 内存配置信息

| 宏名 | 含义 |
| --- | --- |
| **NUMBER_OF_DIMMS_PER_CHANNEL** | 每个 Channel 的 DIMM 插槽数量 |
| **NUMBER_OF_CHANNEL_PER_SOCKET** | 每个 Socket 的内存通道数量 |
| **BLDCFG_I2C_MUX_ADDRESS** | I2C mux 的地址 |
| **BLDCF_SPD_CH_#_DIMM#_ADDRESS** | DIMM 插槽的 I2C 地址 |

### 5.2.2 SMBIOS Table Type 17

内存信息最终会在被存储于 SMBIOS Table Type 17 中，系统启动后用户将可以在 BIOS Setup 下看到内存信息或者通过任何可以读取 SMBIOS 表的工具来获取表中的内存信息。

因此，为了适配新的主板，BIOS 需要修改 Type 17 密切相关的两张 Channel 映射表：HostToApcbChanXLatTab 和 ApcbToBoardChanXLatTab。这两张表分别定义了逻辑 Channel 与物理 Channel 的对应关系和物理 Channel 与主板丝印层 Channel 的映射关系，两张表共用同一个数据结构：

```
typedef struct _HOST_TO_APCB_CHANNEL_XLAT {
  UINT8   RequestedChannelId;
  UINT8   TranslatedChannelId;
} HOST_TO_APCB_CHANNEL_XLAT;
```

1) 在HostToApcbChanXLatTab表中RequestedChannelId对应逻辑Channel, TranslatedChannelId对应物理Channel。这张表的初始化位于：(AgesaModulePkg\Mem\AmdMemChanXLatZpPei\MemChanXLatZpPei.c),之后会被发布为一个Ppi，并与gAmdMemChanXlatPpiGuid绑定，这部分OEM厂商请勿修改。
2) 在ApcbToBoardChanXLatTab表中RequestedChannelId对应物理Channel, TranslatedChannelId对应主板丝印层的Channel。可通过PCD: PcdApcbToBoardChanXLatTab来配置，需要注意的是，ApcbToBoardChanXLatTab中的RequestedChannelId序列要与HostToApcbChanXLatTab中的TranslatedChannelId序列保持一致。

以 Hygon52D16 平台为例，以上两张表的定义分别如下：

HostToApcbChanXLatTab：

```
HOST_TO_APCB_CHANNEL_XLAT SL1r2ChannelXlatTable[] = {
  // Channel A = 0, Channel B = 1, Channel C = 2, Channel D = 3
  // Channel E = 4, Channel F = 5, Channel G = 6, Channel H = 7
  // Requested   Translated
  { 0,        3 },
  { 1,        2 },
  { 2,        7 },
  { 3,        6 },
  { 0xFF,     0xFF},
};
```

PcdApcbToBoardChanXLatTab：

{0x3,0x1, 0x2,0x0, 0x7,0x3, 0x6,0x2}。

以上两张表之间的关系如图 5-1 所示：

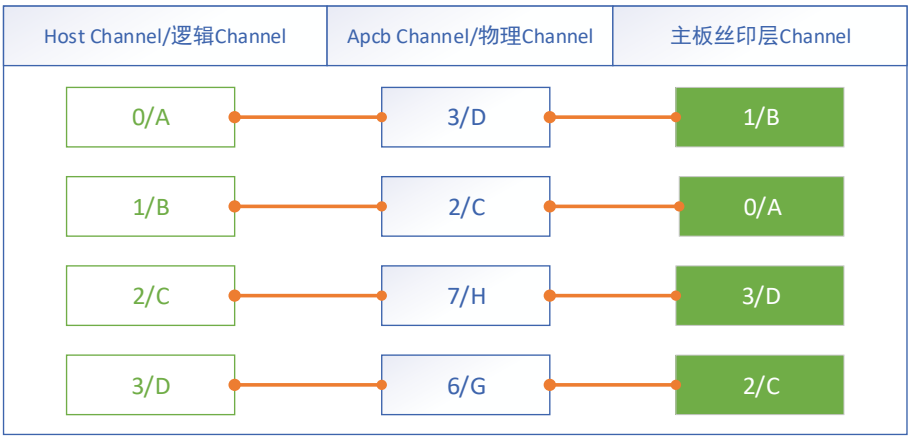



图 5-1 Channel 映射关系

用户只需要配置PcdApcbToBoardChanXLatTab即可，而APCB Channel与主板丝印层Channel对应关系需要在主板原理图中体现，其分别为图 5-2红框和绿框标注：

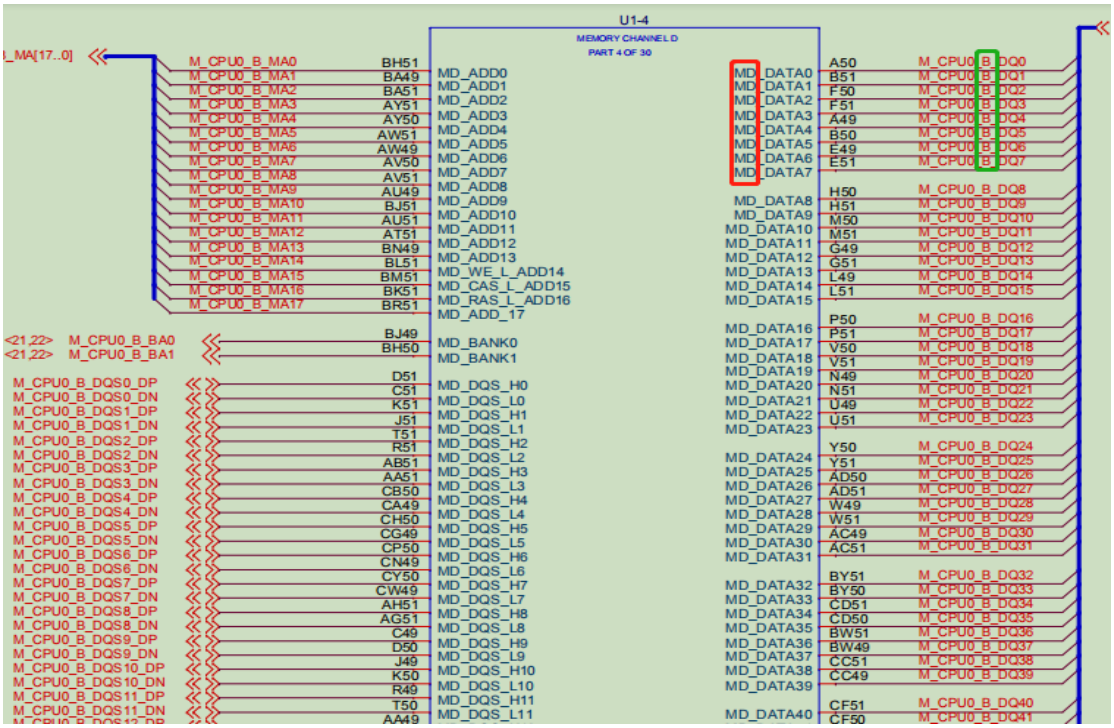



图 5-2 主板原理图物理 Channel 与主板丝印 Channel 的对应关系

### 5.2.3 Silkscreen

内存插槽的丝印字符串可以通过 PcdAmdSmbiosT17Socket#Channel#Dimm##Locator 进行配置。

## 5.3 GPIO

在 AmdCpmOemTable.c 中修改结构体数组 gCpmGpioInitTable<Platform>，可以对主板的 GPIO 用法进行定义。

## 5.4 DXIO

在 AmdCpmOemTable.c 中修改结构体 gCpmDxioTopologyTable<Platform>S#，可以对主板的 DXIO 布局进行定义。

宏 DXIO_ENGINE_DATA_INITIALIZER 用来定义 Lane 分组，其参数按顺序列于下表。

表格 5-3 DXIO Lane 分组定义参数

| 参数 | 含义 |
|---|---|
| **mType** | 该组 Lane 的用途，用作 PCIe 还是 SATA。 |
| **mStartLane** | 该组 Lane 的起始 Lane 和结束 Lane 的编号。顺序可以交换。Lane Mapping 如表格 5-4 所示。 |
| **mEndLane** | |
| **mHotPlug** | 启用或禁用热插拔（只适用于 PCIe） |
| **mGpioGroupId** | 固定为 1 |

表格 5-4 Lane ID Mappping

| Die | Lanes | Package Group | PCIe controller | BIOS Mapping |
|---|---|---|---|---|
| **0** | 00-15 | P0 | PCIe0(TYPE A) | 00-15 |
| **0** | 16-31 | G0 | PCIe1(TYPE B) | 16-31 |
| **1** | 00-15 | P1 | PCIe0(TYPE A) | 32-47 |
| **1** | 16-31 | G1 | PCIe1(TYPE B) | 48-63 |
| **2** | 00-15 | P2 | PCIe0(TYPE A) | 64-79 |
| **2** | 16-31 | G2 | PCIe1(TYPE B) | 80-95 |
| **3** | 00-15 | P3 | PCIe0(TYPE A) | 96-111 |
| **3** | 16-31 | G3 | PCIe1(TYPE B) | 112-127 |

宏 DXIO_PORT_DATA_INITIALIZER_<TYPE>用来对 Lane 分组的属性进行设置。具体属性根据分组的类型有所不同，下面分别介绍。

### 5.4.1 PCIe

图 5-3 所示的代码定义了一个宽度为 x16 的 PCIe Slot，Lane 范围是 0~15，对应了 Die0 的 Type A；禁用了热插拔。

DXIO_PORT_DATA_INITIALIZER_PCIE 的参数说明按顺序列于表格 5-5 中。

表格 5-5 DXIO_PORT_DATA_INITIALIZER_PCIE 参数说明

| 参数 | 含义 |
|---|---|
| **mPortPresent** | 启用或禁用该 Lane 分组定义 |
| **mDevAddress** | 请求的 Device 号和 Function 号用来将该 PCIe Port 映射到特定的逻辑 Bridge。 |
| **mDevFunction** | |
| **mHotplug** | |
| **mMaxLinkSpeed** | 设定 POST 阶段最高速率 |
| **mMaxLinkCap** | 限制 Port 的最高速率 |
| **mAspm** | 启用或禁用 ASPM |
| **mAspmL1_1** | 启用或禁用 ASPM L1.1 |
| **mAspmL1_2** | 启用或禁用 ASPM L1.2 |
| **mClkPmSupport** | 启用或禁用 Clock PM |

```
{ // P0 - x16 slot
  0,
  DXIO_ENGINE_DATA_INITIALIZER (DxioPcieEngine, 0, 15, DxioHotplugDisabled, 1),
  DXIO_PORT_DATA_INITIALIZER_PCIE (
    DxioPortEnabled,                    // Port Present
    0,                                  // Requested Device
    0,                                  // Requested Function
    DxioHotplugDisabled,                // Hotplug
    DxioGenMaxSupported,                // Max Link Speed
    DxioGenMaxSupported,                // Max Link Capability
    DxioAspmL0sL1,                      // ASPM
    DxioAspmDisabled,                   // ASPM L1.1 disabled
    DxioAspmDisabled,                   // ASPM L1.2 disabled
    DxioClkPmSupportDisabled            // Clock PM
  )
},
```

图 5-3 DXIO PCIe 配置示例

如果需要对 PCIe 的 TXEQ 进行调节，可以使用如图 5-4 所示代码，放在 DXIO_PORT_DATA_INITIALIZER_PCIE 的后面即可。

```
PHY_PARAMS_START
    PHY_PARAM (GEN3_txX_eq_main, 0x27),
    PHY_PARAM (GEN3_txX_eq_post, 0x4),
    PHY_PARAM (GEN3_txX_eq_pre, 0x0),
    PHY_PARAM (GEN3_txX_vboost_en, 1),
PHY_PARAMS_END
```

图 5-4 DXIO PCIe TXEQ 配置示例

### 5.4.2 SATA

图 5-5 所示的代码将 Die0 的 Lane 0 和 Lane 1 配置为 SATA。需要特别指出的是如果连续的多个 Lane 被配置成 SATA，可以合起来使用一个定义，如图 5-6 所示。

```
{ // S1P0 - SATA
  0,
  DXIO_ENGINE_DATA_INITIALIZER (DxioSATAEngine, 0, 0, DxioHotplugDisabled, 1),
  DXIO_PORT_DATA_INITIALIZER_SATA (
    DxioPortEnabled                     // Port Present
  )
},
{ // S1P0 - SATA
  0,
  DXIO_ENGINE_DATA_INITIALIZER (DxioSATAEngine, 1, 1, DxioHotplugDisabled, 1),
  DXIO_PORT_DATA_INITIALIZER_SATA (
    DxioPortEnabled                     // Port Present
  )
},
```

图 5-5 DXIO SATA 单 Lane 配置示例

```
{ // S1P0 - SATA
  0,
  DXIO_ENGINE_DATA_INITIALIZER (DxioSATAEngine, 0, 2, DxioHotplugDisabled, 1),
  DXIO_PORT_DATA_INITIALIZER_SATA (
    DxioPortEnabled                     // Port Present
  )
},
```

图 5-6 DXIO SATA 多 Lane 配置示例

### 5.4.3 xGMI

xGMI 不需要 DXIO 中特别配置，只需要将相应的 Lane 跳过即可。

### 5.4.4 Unused Lanes

建议将未使用的 Lane 设定为 PCIe，并使用 DXIO_PORT_DATA_INITIALIZER_PCIE 将其禁用。

## 5.5 USB

SL1 芯片有 4 个 die， die 0 和 die1 各有一个 XHCI 控制器，每个控制器支持 2 个 USB 2.0 port 和 2 个 USB 3.0 port。

SL1R2 芯片有 2 个 die， die 0 和 die1 各有一个 XHCI 控制器，每个控制器支持 4 个 USB 2.0 port 和 4 个 USB 3.0 port。

DM1 芯片有 1 个 die，die 0 有一个 XHCI 控制器，每个控制器支持 4 个 USB 2.0 port 和 4 个 USB 3.0 port。

### 5.5.1 XHCI Controller Enable/Disable

如果某个 die 上的所有 USB port 都没有使用，那 BIOS 可以禁用这个 XHCI 控制器。

在 AMI BIOS 代码里可以直接通过修改 Token “HideDieUSBHC”来禁用某个 die 上的 XHCI 控制器。

### 5.5.2 USB OC Configuration

#### 5.5.2.1 配置 OC 触发方式

根据主板上 USB slot 上的 OC 检测芯片的特性，配置各个 die 的 SMN 寄存器 0x16D80130：BIT8=1 低电平触发 OC event，BIT8=0 高电平触发 OC event（默认值）。

#### 5.5.2.2 为 USB port 设置 OC PIN 脚

每个 USB port 可以设置一个 OC 检测 PIN，当主板上的 OC 芯片检测到过流就会拉低或者拉高这个引脚，XHCI 控制器根据映射关系就知道是哪个 port 发生了过流事件。

BIOS 中通过 PcdDieNumUsbPortAndOCPinMap 来指定 USB Port 关联的 OC Pin，这个 Pcd 用于定义 OC Pins 和 Usb Ports 之间的映射关系。例如：

{0x00,0x0, 0x01,0x1, 0x02,0x2, …, 0xFF}

在上述示例中，按照顺序每两个数据为一组，{0x00,0x0,…}这两个数据中的第一个数据高 4 位代表 Die Num,低四位代表从这个 Die 引出的 USB Port 的编号；第二个数据表示与该 USB Port 所绑定的 OC Pin。

## 5.6 Data Fabric

### 5.6.1 MMIO and IO Resource

Dhyana SOC 每个 die 一共有 8 组 IO Base/limit 寄存器：

- **DF::X86IOBaseAddress**

  IOBase 指定 IO base address

- **DF::X86IOLimitAddress**
  IOLimit 指定 IO limit address
  DstFabricId[7]指定 Socket，DstFabricId[6：5]指定 die number

每一组寄存器决定了一个 die 能够使用的 IO 范围，这个 die 上的所有 devices 使用的 IO 资源都必须在这个范围内。

Dhyana SOC 每个 die 一共有 16 组 MMIO Base/limit 寄存器：

- **DF::MmioBaseAddress**
  MmioBaseAddr 指定 MMIO base address
- **DF:: MmioLimitAddress**
  MmioLimitAddr 指定 MMIO Limit address
- **DF:: MmioAddressControl**
  DstFabricId[7]指定 Socket，DstFabricId[6：5]指定 die number

每两组寄存器（一组指定 4G 以下，一组指定 4G 以上）指定一个 die 能够使用的 MMIO 范围，这个 die 上的所有 devices 使用的 MMIO 资源都必须在这个范围内。Die 与 die 之间的 MMIO 和 IO resource 不能冲突。

AGESA PI 在 007 版本导入了统一初始化和分配 MMIO 和 IO resource 的功能函数，包含：

- FabricResourceInit
  初始化 DF 的 IO 和 MMIO base/limit 寄存器，为每个 die 设置 IO 和 MMIO 范围。
- FABRIC_RESOURCE_MANAGER_PROTOCOL
  提供 Protocol 给需要申请 IO 和 MMIO 资源的模块调用，会从 FabricResourceInit 设置好的 IO 和 MMIO 范围来分配资源。

#### 5.6.1.1 FabricResourceInit

FabricResourceInit 函数在 PEI 阶段被调用，读取 PcdAmdFabricResourceDefaultMap 来判断分配 IO 和 MMIO 空间的策略：

值 0，默认值，读取"ResourceSizeForEachDie" variable 的值来决定给每个 die 分多大空间
　　如果获取 variable 失败，就平均分配

值 1，平均分配

BIOS 分配 MMIO 空间有两个策略：

1. **FabricZenZpInitMmioEqually** 平均分配
2. **FabricZenZpInitMmioBaseOnNvVariable** 按照 Variable 设置分配

#### 5.6.1.2 FABRIC_RESOURCE_MANAGER_PROTOCOL

这个 protocol 一共提供了 6 个函数给其它模块使用，统一管理各个 die 上的 MMIO 和 IO 资源，避免用户自己操作产生地址冲突。

```
STATIC                          FABRIC_RESOURCE_MANAGER_PROTOCOL
```

```
mAmdFabricZenZpResourceManager = {
  AMD_FABRIC_RESOURCE_PROTOCOL_REV,
  FabricZenZpAllocateMmio,
  FabricZenZpAllocateIo,
  FabricZenZpGetAvailableResource,
  FabricZenZpReallocateResourceForEachDie,
  FabricZenZpResourceRestoreDefault,
  FabricZenZpEnableVgaMmio
};
```

当用户想为某个 die 上的各种设备分配 MMIO32 和 MMIO64 资源时，只需要调用 FabricZenZpAllocateMmio 函数即可。

当用户想为某个 die 上的各种设备分配 IO 资源时，只需要调用 FabricZenZpAllocateIo 函数即可。

当用户想知道某个 die 剩余的可用资源的 size 时，只需要调用 FabricZenZpGetAvailableResource 函数即可。

当用户觉得目前给每个 die 分配的资源不合理或者不满足使用需求时，可以调用 FabricZenZpReallocateResourceForEachDie 按自己的意愿指定每个 die 各种资源的 length 以及 Alignment，然后重启系统，BIOS 就会按照需求重新分配资源。

当用户不想指定资源分配方式，可以调用 FabricZenZpResourceRestoreDefault 清除 Variable 里保存的分配法则，重启系统后，BIOS 就会重新平均分配资源。

当用户在某个 die 连接的 PCIE 显卡想使用 0xA0000-0xBFFFF 的视频缓冲区，就可以调用 FabricZenZpEnableVgaMmio 来使能这个功能。

- **FabricAllocateMmio**

待 PEI 阶段执行 FabricZenZpInitMmioEqually 或者

FabricZenZpInitMmioBaseOnNvVariable 为每个 die 都分配好 MMIO 后，其它需要申请 MMIO 资源的模块就可以调用 FabricAllocateMmio 函数来从分好的资源里来划分资源了。这个函数会更新每个 die 的 MMIO 资源的使用情况。

```
EFI_STATUS
FabricAllocateMmio (
  IN OUT   UINT64 *BaseAddress,
  IN OUT   UINT64 *Length,
  IN       UINT64                 Alignment,
  IN       FABRIC_TARGET           Target,
  IN OUT   FABRIC_MMIO_ATTRIBUTE *Attributes
  )
```

这个函数的参数如下：

| 名称 | 值 | 方向 | 描述 |
|---|---|---|---|
| BaseAddress | | Out | 分配地址后返回给调用者 |
| Length | | IN | 指定想要分配的内存长度 |
| Alignment | SIZE_1KB<br>SIZE_2KB<br>… | IN | 指定对齐方式 |
| TgtType | TARGET_PCI_BUS | IN | 根据 PciBusNum 来找到所属的 socket 和 die number，然后再分配 MMIO 空间 |
| | TARGET_DIE | IN | 根据 SocketNum 和 DieNum 来分配 MMIO 空间 |
| Attributes | MMIO_BELOW_4G<br>MMIO_ABOVE_4G<br>P_MMIO_BELOW_4G<br>P_MMIO_ABOVE_4G<br>NON_PCI_DEVICE_BELOW_4G | IN | 指定 MmioType，Below 表示 4G 以下，ABOVE 表示 4G 以上；P 表示 Prefetch，没有 P 表示 Non-prefetch；<br>NON_PCI_DEVICE_BELOW 表示为 FCH，PSP，IOAPIC，IOMMU 这种非 PCI 枚举设备分配资源 |

```
/// DF target
typedef struct _FABRIC_TARGET {
  UINT16  PciBusNum:8;          ///< PCI bus number
  UINT16  SocketNum:2;          ///< Socket number
  UINT16  DieNum:5;             ///< Die number
  UINT16  TgtType:1;            ///< Indicator target type
                              ///< 0 - TARGET_PCI_BUS - set up an MMIO
region for the device on a certain PCI bus
                              ///< 1 - TARGET_DIE     - set up an MMIO
region for the device on a certain Socket, DIE
} FABRIC_TARGET;
```

```
/// MMIO attribute
typedef struct _FABRIC_MMIO_ATTRIBUTE {
  UINT8    ReadEnable:1;          ///< Indicator whether the range is
readable
  UINT8    WriteEnable:1;         ///< Indicator whether the range is
writable
  UINT8    NonPosted:1;           ///< Indicator whether the range is
posted
  UINT8    CpuDis:1;
  UINT8    :1;                  ///< Reserved
  UINT8    MmioType:3;           ///< We have 5 pools per Die.
0) MMIO_BELOW_4G
1) MMIO_ABOVE_4G
2) P_MMIO_BELOW_4G
3) P_MMIO_ABOVE_4G
4) NON_PCI_DEVICE_BELOW_4G
} FABRIC_MMIO_ATTRIBUTE;
```

举例 1：在 PEI 阶段为某个 die 分配 4GB 以下的 Non- PCI MMIO 空间

```
  UINT64     Length;
  UINT64     FchMmioBase;
  Length = 0x2000;
  MmioTarget.TgtType = TARGET_DIE;
  MmioTarget.SocketNum = Die / MAX_DIES_PER_SOCKET;
  MmioTarget.DieNum = Die % MAX_DIES_PER_SOCKET;
  Attributes.ReadEnable = 1;
  Attributes.WriteEnable = 1;
  Attributes.NonPosted = 0;
  Attributes.MmioType = NON_PCI_DEVICE_BELOW_4G;
  FabricAllocateMmio (&FchMmioBase, &Length, 16, MmioTarget, &Attributes)
```

举例 2：在 BDS PCI 枚举阶段为某个 die 分配 32 位的 Non – Prefetch MMIO 空间

```
UINT64                   RbMmioActualBase = 0xffffffff;
FABRIC_TARGET             MmioTarget;
FABRIC_MMIO_ATTRIBUTE      Attributes;
UINT64                   Length = 0x200000;

MmioTarget.TgtType = TARGET_PCI_BUS;
MmioTarget.SocketNum = 0;
MmioTarget.DieNum = 0;
MmioTarget.PciBusNum = 0x20;
Attributes.ReadEnable = 1;
```

```
Attributes.WriteEnable = 1;
Attributes.NonPosted = 0;
Attributes.MmioType = MMIO_BELOW_4G;
RbMmioActualBase  = 0;

Status = FabricResourceManager->FabricAllocateMmio (
           FabricResourceManager,
           &RbMmioActualBase,
           &Length,
           0xFFFFF,
           MmioTarget,
           &Attributes);
```

举例 3：在 BDS PCI 枚举阶段为某个 die 分配 64 位的 Non – Prefetch MMIO 空间

```
UINT64                   RbMmioActualBase = 0xffffffff;
FABRIC_TARGET             MmioTarget;
FABRIC_MMIO_ATTRIBUTE      Attributes;
UINT64                   Length = 0x200000;

MmioTarget.TgtType = TARGET_PCI_BUS;
MmioTarget.SocketNum = 0;
MmioTarget.DieNum = 0;
MmioTarget.PciBusNum = 0x20;
Attributes.ReadEnable = 1;
Attributes.WriteEnable = 1;
Attributes.NonPosted = 0;
Attributes.MmioType = MMIO_ABOVE_4G;
RbMmioActualBase  = 0;

Status = FabricResourceManager->FabricAllocateMmio (
           FabricResourceManager,
           &RbMmioActualBase,
           &Length,
           0xFFFFF,
           MmioTarget,
           &Attributes);
```

- **FabricAllocateIo**

```
EFI_STATUS
EFIAPI
FabricZenZpAllocateIo (
  IN      FABRIC_RESOURCE_MANAGER_PROTOCOL *This,
  IN OUT   UINT32                   *BaseAddress,
```

```
  IN OUT   UINT32                    *Length,
  IN       FABRIC_TARGET              Target
  )
```

这个函数和 FabricAllocateMmio 一样，设置 Target 参数时既可以通过 bus number 来查找 die，也可以直接指定 socket 和 die number。
需要的 IO 长度也是由 length 参数指定，按照从高地址到低地址的分配原则，分配的 base 在 base address 参数返回。
分配完后，会自动更新 FABRIC_IO_REGION 的 IoUsed 值。
注意：IO 分配不需要指定对齐方式，默认 0xFFF，4KB 对齐。

举例 1：在 BDS PCI 枚举阶段为某个 die 分配一段 IO 空间

```
UINT32          RbIoBase = 0xffff;
FABRIC_TARGET   IoTarget;
UINT32          Length = 0x1000;

IoTarget.TgtType = TARGET_PCI_BUS;
IoTarget.SocketNum = 0;
IoTarget.DieNum = 0;
IoTarget.PciBusNum = 0x20;
RbIoBase  = 0;

Status = FabricResourceManager->FabricAllocateIo (
             FabricResourceManager,
             &RbIoBase,
             &Length,
             IoTarget);
```

- **FabricGetAvailableResource**

```
EFI_STATUS
EFIAPI
FabricZenZpGetAvailableResource (
  IN       FABRIC_RESOURCE_MANAGER_PROTOCOL  *This,
  IN                                FABRIC_RESOURCE_FOR_EACH_DIE
*ResourceForEachDie
  )
```

用户调用这个函数可以获取到当前空间可用的 MMIO 和 IO size，结果保存在 ResourceForEachDie 指针。

```
typedef struct _FABRIC_ADDR_APERTURE {
```

```
  UINT64  Base;
  UINT64  Size;
  UINT64  Alignment;      //< Alignment bit map. 0xFFFFF
means 1MB align
} FABRIC_ADDR_APERTURE;

/// Resource for each Die
typedef struct _FABRIC_RESOURCE_FOR_EACH_DIE {
  FABRIC_ADDR_APERTURE
NonPrefetchableMmioSizeAbove4G[MAX_SOCKETS_SUPPORTED][MA
X_DIES_PER_SOCKET];
  FABRIC_ADDR_APERTURE
PrefetchableMmioSizeAbove4G[MAX_SOCKETS_SUPPORTED][MAX_D
IES_PER_SOCKET];
  FABRIC_ADDR_APERTURE
NonPrefetchableMmioSizeBelow4G[MAX_SOCKETS_SUPPORTED][MA
X_DIES_PER_SOCKET];
  FABRIC_ADDR_APERTURE
PrefetchableMmioSizeBelow4G[MAX_SOCKETS_SUPPORTED][MAX_D
IES_PER_SOCKET];
  FABRIC_ADDR_APERTURE
Die0SecondNonPrefetchableMmioSizeBelow4G;
  FABRIC_ADDR_APERTURE
Die0SecondPrefetchableMmioSizeBelow4G;
  FABRIC_ADDR_APERTURE
IO[MAX_SOCKETS_SUPPORTED][MAX_DIES_PER_SOCKET];
} FABRIC_RESOURCE_FOR_EACH_DIE;
```

BIOS 在 PCI 枚举时为各个 die 的 PCI 设备分配资源之前，可以先获取 available 值和需要的资源长度比较一下，如果不满足需求，就需要调用 FabricReallocateResourceForEachDie 重新设置各个 die 的 resource size。

举例：

```
EFI_STATUS                      Status = EFI_SUCCESS;
FABRIC_RESOURCE_FOR_EACH_DIE       AvailResource;
FABRIC_RESOURCE_FOR_EACH_DIE       RequestResource;

Status                                                  =
FabricResourceManager->FabricGetAvailableResource(FabricResourceMan
ager, &AvailResource);
if (EFI_ERROR (Status)){
       return EFI_NOT_READY;
}

MmioLength = 0x3800000;
```

```
if                              (MmioLength                              >
AvailResource.NonPrefetchableMmioSizeBelow4G[SocketNum][DieNum].Siz
e) {
    NeedReallocateMmio32Resource = TRUE;
}
```

- **FabricReallocateResourceForEachDie**

```
EFI_STATUS
EFIAPI
FabricZenZpReallocateResourceForEachDie (
  IN      FABRIC_RESOURCE_MANAGER_PROTOCOL  *This,
  IN                          FABRIC_RESOURCE_FOR_EACH_DIE
*ResourceSizeForEachDie,
  IN      FABRIC_ADDR_SPACE_SIZE           *SpaceStatus
  )
```

ResourceSizeForEachDie 指定用户想为每个 die 分配的各种资源的长度，

Alignment。

SpaceStatus 设置成 NULL；
这个函数仅把用户请求的这些值写到"ResourceSizeForEachDie" variable，执行完这个函数，用户需要重启系统，重启后 PEI 阶段 FabricResourceInit 函数才会按照 variable 的值重设 DF 寄存器。
举例：

```
FABRIC_RESOURCE_FOR_EACH_DIE       RequestResource;
UINT8                          SocketNum, DieNum;
SocketNum = 0;
DieNum = 1;
RequestResource.NonPrefetchableMmioSizeBelow4G[SocketNum][DieNum].S
ize = 0x5000000;
RequestResource.NonPrefetchableMmioSizeBelow4G[SocketNum][DieNum].A
lignment = 0xFFFFF;
RequestResource.PrefetchableMmioSizeBelow4G[SocketNum][DieNum].Size
= 0x200000;
RequestResource.PrefetchableMmioSizeBelow4G[SocketNum][DieNum].Alig
nment = 0xFFFFF;
RequestResource.IO[SocketNum][DieNum].Size = 0x2000;
RequestResource.IO[SocketNum][DieNum].Alignment = 0xFFF;

Status = FabricResourceManager->FabricReallocateResourceForEachDie(
            FabricResourceManager,
            &RequestResource,
```

```
            NULL);

pRS->ResetSystem(EfiResetWarm, EFI_SUCCESS, 0, NULL);
```

- **FabricResourceRestoreDefault**

```
EFI_STATUS
EFIAPI
FabricZenZpResourceRestoreDefault (
  IN       FABRIC_RESOURCE_MANAGER_PROTOCOL     *This

  )
```

这个函数没有参数，作用是清除"ResourceSizeForEachDie" variable，即下一次重启后，BIOS 会在 FabricResourceInit 函数按照 die 的数量平均分配 MMIO 和 IO 资源。

- **FabricEnableVgaMmio**

```
EFI_STATUS
EFIAPI
FabricZenZpEnableVgaMmio (
  IN       FABRIC_RESOURCE_MANAGER_PROTOCOL  *This,
  IN       FABRIC_TARGET          Target
  )
```

这个函数的参数 Target 与 FabricAllocateMmio 函数里的 Target 含义相同，作用是把 MMIO 空间 0xA0000-0xBFFFF decode 到这个 die，即把 DF 上的 VGAEn 寄存器的 BIT0 置 1，BIT2 清 0。

如果主板上的显卡使用的是 legacy VBIOS，就需要做这一步。

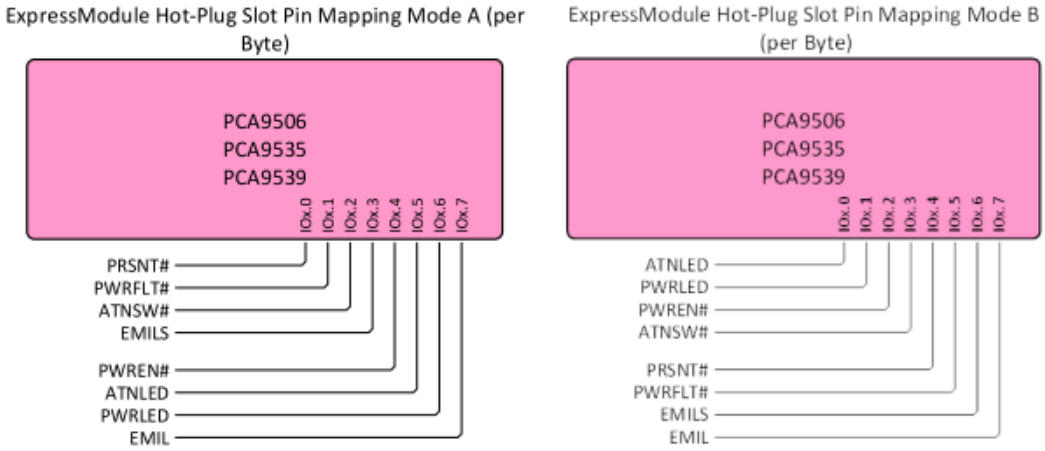
## 5.7 使能 Hotplug 功能

- 在 AmdCpmOemtable.c 中将 PCIE 插槽设置为 hotpulug 属性

```
{ // P0 - x4 PCIEx4
    0,
    DXIO_ENGINE_DATA_INITIALIZER (DxioPcieEngine, 8, 11, DxioHotplugServerExpress, 1),
    DXIO_PORT_DATA_INITIALIZER_PCIE (
    DxioPortEnabled,                    // Port Present
    0,                                  // Requested Device
    0,                                  // Requested Function
    DxioHotplugServerExpress,               // Hotplug
    DxioGenMaxSupported,                // Max Link Speed
    DxioGenMaxSupported,                // Max Link Capability
    DxioAspmL0sL1,                      // ASPM
    DxioAspmDisabled,                   // ASPM L1.1 disabled
    DxioAspmDisabled,                   // ASPM L1.2 disabled
    DxioClkPmSupportDisabled            // Clock PM
    )
  },
```

- 配置 HOTPLUG_DESCRIPTOR，start lane 和 end lane 为在 Dxiotable 中配置支持热插拔的 slot 所占用的 lane 号。Socket 为所在的 socket 号（第一个 socket 为 0，第二个位 1）。Slot 号需要和 porting 主板的时候使用的 slot 号对应。

HotPlug Type 为选择 PCA9555A 引脚的连接方式，现在可以支持 2 中引脚定义：




当使用 Mode A 的时候，HotPlug Type 配置为 HotplugExpressModule，当使用 Mode B 的时候配置为：HotplugExpressModuleB

示例如下：

```
  {
      0,
      HOTPLUG_ENGINE_DATA_INITIALIZER (32,     // Start Lane (J18)
                                       35,     // End Lane
                                       0,      // Socket Number
                                       18)      // Slot number
      PCIE_HOTPLUG_INITIALIZER_MAPPING (HotplugExpressModuleB,   //Hotplug Type
                                        0,     // 0 = No Gpio Descriptor
                                        0,     // 0 = No Reset Descriptor
                                        1,     // 1 = Port Active - this is a valid
descriptor
                                        1,     // 1 = Master/Slave APU
                                        1,     // 0 = Die number this slot is connected
to
                                        0,     // Alternate Slot number
                                        0)     // Primary/secondary for SSD only
      PCIE_HOTPLUG_INITIALIZER_FUNCTION (4,    // Gpio Bit Select
                                         0,    // GPIO Byte Mapping
                                         0,    // GPIO Device Mapping Ext
                                         0,    // GPIO Device Mapping
                                         1,    // Device Type 1 = 9535
                                         7,    // Bus Segment (No 9545 Present)
                                         0x08)    // Function Mask  //AMD fix for NVMe
      PCIE_HOTPLUG_INITIALIZER_NO_RESET()
      PCIE_HOTPLUG_INITIALIZER_NO_GPIO()
  },
```