Gmac driver implementation under UEFI

原创 Andy Lau Haidian Branch Modified on 2022-02-28 16:20:25 Read 2.2k Collection 12 Likes 1

摘要 This article explains in detail the classification of Mac controllers, especially Gmac in data stream filtering, working mode (half-duplex/full-duplex), and the installation and initialization process of Gmac drivers in UEFI, including key steps in support functions and Start functions. In addition, the article deeply analyzes the hardware initialization and configuration process in the GmacFirstTimeInit function.

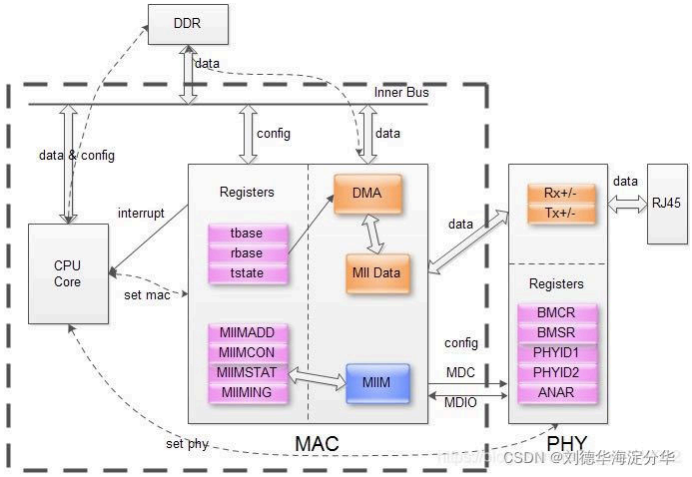
The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

1. Classification of mac:
mac can be divided into emac and gmac according to the transmission rate.

AI generated projects 登录复制

```
mac: 它是一个controller,它的主要的作用有两个方面:
     帧发送: 接受来自协议层的数据,加上控制信息,然后以位数据流的形式传到物理层.
     帧接受: 接受物理层的数据流,检查是否有效,然后发送给上层协议,或者丢弃.
     所以从上述描述来看,gmac控制器主要完成的工作是数据流的过滤

phy: gmac 有对应的phy, 而phy是实际上操作数据的收发的.
```



2. Mac working mode:

AI generated projects 登录复制

```
工作模式可以分为半双工和全双工
半双工:
        GMAC Client 将数据传送给Gmac后,Gmac先给数据加上Preamble, FSD,FCS组成以太网帧;
        然后检查载波侦听信号 (CRS) 若有载波信号,表示有数据正在在本地网段上传播,就等待直到载波信号消失.
        载波信号消失后,Gmac还要等待一个帧间延时,若在帧间延时期间,一直没有载波信号,该以太网帧就可以
        开始向物理层传输数据.

全双工:
        全双工模式: Gmac从Gmac Client 接收到数据后,不需要载波帧听,直接向物理层传递,
        其它操作与半双工相同.
```

AI Assistant

收起 ^

3. Read and write the registers of the Phy chip through the following two registers to complete the control of the phy

9.11.5.15. GMAC MII Command Register(Default Value: 0x0000_0000)

Offset: 0x0048			Register Name: GMAC_MII_CMD
Bit	**Read/Write**	**Default/Hex**	**Description**
31:23	/	/	/
22:20	R/W	0x0	MDC_DIV_RATIO_M MDC Clock Divide Ratio 000: 16 001: 32 010: 64 011: 128 Others: Reserved **Note:** MDC Clock is divided from AHB clock
19:17	/	/	/
16:12	R/W	0x0	PHY_ADDR PHY Address
11:9	/	/	/
8:4	R/W	0x0	PHY_REG_ADDR PHY Register Address
3:2	/	/	/
1	R/W	0x0	MII_WR MII Write and Read 0: Read 1: Write
0	R/W	0x0	MII_BUSY 0: Write no valid, read 0 indicate finish in read or write operation 1: Write start read or write operation, read 1 indicate busy.

9.11.5.16. GMAC MII Data Register(Default Value: 0x0000_0000)

Offset: 0x004C			Register Name: GMAC_MII_DATA
Bit	**Read/Write**	**Default/Hex**	**Description**
31:16	/	/	/
15:0	R/W	0x0	MII_DATA Written to or read from the register in the selected PHY.

4. Implementation of Gmac driver in Uefi:

c AI generated projects 登录复制 run

```c
EFI_STATUS
EFIAPI
InitializeGmacUNDIDriver (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE *SystemTable
  )
{
  if(NetworkProtocolIsDisable())
    return EFI_SUCCESS;

  Status = EfiLibInstallDriverBinding (ImageHandle, SystemTable,
                                       &gUndiDriverBinding, ImageHandle);

  Status = gBS->InstallMultipleProtocolInterfaces (
                 &gUndiDriverBinding.DriverBindingHandle,
                 &gEfiComponentNameProtocolGuid,
                 &gUndiComponentName,

```

AI Assistant

```
                &gEfiDriverDiagnosticsProtocolGuid,
                &gGmacUndiDriverDiagnostics,
                &gEfiComponentName2ProtocolGuid,
                &gUndiComponentName2,
                &gEfiDriverDiagnostics2ProtocolGuid,
                &gGmacUndiDriverDiagnostics2,
                &gEfiDriverConfigurationProtocolGuid,
                &gGmacUndiDriverConfiguration,
                &gEfiDriverHealthProtocolGuid,
                &gUndiDriverHealthProtocol,
                NULL
              );
  // This protocol does not need to be closed because it uses the GET_PROTOCOL attribute
  Status = gBS->OpenProtocol (
                ImageHandle,
                &gEfiLoadedImageProtocolGuid,
                (VOID *) &LoadedImageInterface,
                ImageHandle,
                NULL,
                EFI_OPEN_PROTOCOL_GET_PROTOCOL
              );

  if (EFI_ERROR (Status)) {
    DEBUGPRINT (CRITICAL, ("OpenProtocol returns %r\n", Status));
    return Status;
  }

  LoadedImageInterface->Unload = GmacUndiUnload;
    Status = gBS->CreateEvent (
                EVT_SIGNAL_EXIT_BOOT_SERVICES,
                TPL_NOTIFY,
                GmacUndiNotifyExitBs,
                NULL,
                &mEventNotifyExitBs
              );
  if (EFI_ERROR (Status)) {
    DEBUGPRINT (CRITICAL, ("CreateEvent returns %r\n", Status));
    return Status;
  }
  Status = InitializePxeStruct ();

  return Status;
}
```

收起 ^

AI generated projects 登录复制

首先是安装gUndiDriverBinding,实例化的时候:

c AI generated projects 登录复制 run

```c
EFI_DRIVER_BINDING_PROTOCOL gUndiDriverBinding = {
  GmacUndiDriverSupported, // Supported
  GmacUndiDriverStart,     // Start
  GmacUndiDriverStop,      // Stop
  VERSION_TO_HEX,         // Driver Version
  NULL,                   // ImageHandle
  NULL                    // Driver Binding Handle
};
```

AI Assistant

AI generated projects 登录复制

首先我们分析一下Support函数:

```c
/**
Test to see if this driver supports ControllerHandle.
**/
EFI_STATUS
EFIAPI
GmacUndiDriverSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *This,
  IN EFI_HANDLE                  Controller,
  IN EFI_DEVICE_PATH_PROTOCOL *  RemainingDevicePath
  )
{
  EFI_STATUS           Status;
  EFI_PCI_IO_PROTOCOL *PciIo;
  PCI_TYPE00           Pci;
  UNDI_PRIVATE_DATA   *UndiPrivateData;

  UndiPrivateData = GetControllerPrivateData (Controller);

  if (UndiPrivateData == NULL) {
    DbgPrint (EFI_D_INFO,"LY_TEST------The UndiPrivateData is NULL---\n");
    Status = gBS->OpenProtocol (
                    Controller,
                    &gEfiPciIoProtocolGuid,
                    (VOID **) &PciIo,
                    This->DriverBindingHandle,
                    Controller,
                    EFI_OPEN_PROTOCOL_BY_DRIVER
                  );

    if (EFI_ERROR (Status)) {
      return Status;
    }
  } else {
    PciIo = UndiPrivateData->NicInfo.PciIo;
    if (PciIo == NULL) {
      return EFI_INVALID_PARAMETER;
    }
  }
  Status = PciIo->Pci.Read (
                       PciIo,
                       EfiPciIoWidthUint8,
                       0,
                       sizeof (PCI_CONFIG_HEADER),
                       &Pci
                     );
  if (EFI_ERROR (Status)) {
    goto ExitSupported;
  }

  if (Status == EFI_SUCCESS) {
    DbgPrint (EFI_D_INFO,"ly_test----Pci.Hdr.VendorId = %lx--Pci.Hdr.DeviceId = %lx-----\n",Pci.Hdr.VendorId,Pci.Hdr.DeviceId);
  }

  if (!IsDeviceIdSupported (Pci.Hdr.VendorId, Pci.Hdr.DeviceId)) {
    Status = EFI_UNSUPPORTED;
    goto ExitSupported;
  }

  Status = AnalyzeRemainingDevicePath (
             UndiPrivateData,
             RemainingDevicePath
           );
  if (EFI_ERROR (Status)) {

```

```
    goto ExitSupported;
  }

ExitSupported:
  if (UndiPrivateData == NULL) {
    gBS->CloseProtocol (
           Controller,
           &gEfiPciIoProtocolGuid,
           This->DriverBindingHandle,
           Controller
         );
  }

  return Status;
}
```

收起 ^

AI generated projects 登录复制

```
Support函数里面需要注意的有两个地方:
1. 首先得到Undi的私有数据
UNDI_PRIVATE_DATA *UndiPrivateData;
UndiPrivateData = GetControllerPrivateData (Controller);
```

c AI generated projects 登录复制 run

```
UNDI_PRIVATE_DATA*
GetControllerPrivateData (
  IN  EFI_HANDLE ControllerHandle
  )
{
  UINT32 i = 0;

  for (i = 0; i < mActiveControllers; i++) {
    if (mGMACUndi32DeviceList[i] != NULL) {
      if (mGMACUndi32DeviceList[i]->ControllerHandle == ControllerHandle) {
        return mGMACUndi32DeviceList[i];
      }
    }
  }
  return NULL;
}

```

收起 ^

AI generated projects 登录复制

```
mGMACUndi32DeviceList 是一个全局变量,类型是UNDI_PRIVATE_DATA *的一个数组.
UNDI_PRIVATE_DATA *mGMACUndi32DeviceList[MAX_NIC_INTERFACES];
我们看一下UNDI_PRIVATE_DATA 的结构:
```

c AI generated projects 登录复制 run

```
typedef struct UNDI_PRIVATE_DATA_S {
  UINTN                                   Signature;
  EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL NiiProtocol31;
  EFI_NII_POINTER_PROTOCOL                NIIPointerProtocol;
  EFI_HANDLE                              ControllerHandle;
  EFI_HANDLE                              DeviceHandle;
  EFI_HANDLE                              HiiInstallHandle;
  EFI_HANDLE                              FmpInstallHandle;
  EFI_DEVICE_PATH_PROTOCOL *              Undi32BaseDevPath;
  EFI_DEVICE_PATH_PROTOCOL *              Undi32DevPath;

```

```
  GIG_DRIVER_DATA                         NicInfo;
  EFI_UNICODE_STRING_TABLE *              ControllerNameTable;
  CHAR16 *                                Brand;
  BOOLEAN                                 IsChildInitialized;
  // HII Configuration
  EFI_HII_HANDLE                          HiiHandle;
  UNDI_DRIVER_CONFIGURATION               Configuration;
  /* HII Configuration parameters start here
   depending on these settings some of HII menus are disabled */
  BOOLEAN                   LinkSpeedSettingsSupported;
  UINT8 AltMacAddrSupported;
  // Consumed protocol
  EFI_HII_DATABASE_PROTOCOL *      HiiDatabase;
  EFI_HII_STRING_PROTOCOL *        HiiString;
  EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting;
  EFI_FORM_BROWSER2_PROTOCOL *     FormBrowser2;
  EFI_GUID                         HiiFormGuid;
  // Produced protocol
  EFI_HII_CONFIG_ACCESS_PROTOCOL ConfigAccess;
  EFI_DRIVER_STOP_PROTOCOL       DriverStop;
  EFI_ADAPTER_INFORMATION_PROTOCOL AdapterInformation;
  UINT32                           LastAttemptVersion;
  UINT32                           LastAttemptStatus;
} UNDI_PRIVATE_DATA;
```

收起 ^

AI generated projects 登录复制

```
到这里,很多朋友会有疑问,UNDI是什么? 以及我们为什么要用UndiPrivateData.
首先解释一下什么是UNDI:
    UNDI的全称是Universal Network Driver Interface
    它并不是UEFI网络框架的一部分，甚至也可以不是UEFI的一部分。
    不过目前UEFI下的网络驱动都会实现UNDI，这样UEFI就可以通过SNP来调用网卡的底层驱动。
    UNDI说到底是定义了一系列的接口，然后SNP来访问这些接口。
    这需要实现了UNDI的网络设备驱动中安装一个NetworkInterfaceIdentifier（NII）协议.
    而Gmac驱动本身来说也是Gmac网卡的驱动,所以实现UNDI来完成对Gmac的调用.
所以在访问PCI的IO空间时,如果UndiPrivateData 为空,那么我们就用驱动所绑定的句柄
上的PciIo去访问,否则的话: PciIo = UndiPrivateData->NicInfo.PciIo;
就用网卡信息中的PciIo去访问IO空间.

2.   if (!IsDeviceIdSupported (Pci.Hdr.VendorId, Pci.Hdr.DeviceId)) {
        Status = EFI_UNSUPPORTED;
        goto ExitSupported;
    }
    检查VendorId和DeviceId,我们龙芯3A571(3A5000+7A1000,胡老师叫3A571,
    哈哈,虽然听着土,但是还是很好理解的哈) 上面的 GMAC 的VerdorId 是0x 0014,
    DeviceId 是 0x7a03.

总结: Support函数主要作用是扫描主板上的所有PCI设备,如果存在
     和Gmac设备匹配的DeviceId和VendorId的话,说明这个驱动是支持控制器
     的句柄的安装的.


接下来再来分析一下 Start函数:
```

收起 ^

c AI ted projects 登录复制 run

```
/** Start this driver on Controller by opening PciIo and DevicePath protocol.

   Initialize PXE structures, create a copy of the Controller Device Path with the
   NIC's MAC address appended to it, install the NetworkInterfaceIdentifier protocol
   on the newly created Device Path.

```

AI Assistant

```
   通过打开PciIo和DevicePath协议在Controller上启动这个驱动程序。
   初始化PXE结构，创建一个带有网卡MAC地址的控制器设备路径的副本，在新创建的设备路径上安装    NetworkInterfaceIdentifier协议。
**/

EFI_STATUS
EFIAPI
GmacUndiDriverStart (
  IN EFI_DRIVER_BINDING_PROTOCOL *This,
  IN EFI_HANDLE                  Controller,
  IN EFI_DEVICE_PATH_PROTOCOL *  RemainingDevicePath
  )
{
  UNDI_PRIVATE_DATA *UndiPrivateData      = NULL;
  EFI_STATUS         Status               = EFI_SUCCESS;
  BOOLEAN            InitializeChild      = TRUE;
  BOOLEAN            InitializeController = TRUE;
  ...
  if (InitializeController) {
    DbgPrint (EFI_D_INFO,"ly_test-----------GmacUndiDriverStart InitializeController---\n");
    Status = InitUndiPrivateData (
               Controller,
               &UndiPrivateData
             );
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("InitUndiPrivateData returns %r\n", Status));
      DEBUGWAIT (CRITICAL);
      goto UndiError;
    }
        Status = OpenControllerProtocols (
               Controller,
               This,
               UndiPrivateData
             );
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("OpenContollerProtocols returns %r\n", Status));
      DEBUGWAIT (CRITICAL);
      goto UndiError;
    }
    Status = InitController (UndiPrivateData);
    if (EFI_ERROR (Status) &&
      (Status != EFI_ACCESS_DENIED))
    {
      DEBUGPRINT (CRITICAL, ("InitController fails: %r", Status));
      goto UndiErrorDeleteDevicePath;
    }
    Status = InitControllerProtocols (
               UndiPrivateData,
               Controller
             );
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("InitControllerProtocols failed with %r\n", Status));
      goto UndiErrorDeleteDevicePath;
    }
  }
  if (InitializeChild) {
    InitUndiStructures (UndiPrivateData);

    Status = InitChild (UndiPrivateData);
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("InitChild failed with %r\n", Status));
      goto UndiErrorDeleteDevicePath;
    }

    Status = InitChildProtocols (
               UndiPrivateData
```

```
            );
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("InitChildProtocols failed with %r\n", Status));
      goto UndiErrorDeleteDevicePath;
    }
    Status = OpenChildProtocols (
               UndiPrivateData,
               This,
               Controller
             );
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("OpenChildProtocols failed with %r\n", Status));
    }
    UndiPrivateData->IsChildInitialized = TRUE;
  }

  Status = gBS->CreateEvent (
                 EVT_TIMER | EVT_NOTIFY_SIGNAL,
                 TPL_NOTIFY,
                 GmacReInit,
                 (UndiPrivateData->NicInfo.GmacHw),
                 &(UndiPrivateData->NicInfo.GmacHw->synopGMACdev->ReInit)
                 );

  if (EFI_ERROR (Status)) {
    ASSERT(0);
    return Status;
  }
  Status = gBS->SetTimer (UndiPrivateData->NicInfo.GmacHw->synopGMACdev->ReInit, TimerPeriodic, (UINT64)CHECK_INIT_PERIOD);
   if (EFI_ERROR (Status)) {
    ASSERT(0);
    return Status;
   }

  return EFI_SUCCESS;

UndiErrorDeleteDevicePath:
  GigUndiPxeUpdate (NULL, mGMACPxe31);
  gBS->FreePool (UndiPrivateData->Undi32DevPath);

UndiError:
  mGMACUndi32DeviceList[mActiveControllers - 1] = NULL;
  mActiveControllers--;

  CloseControllerProtocols (
    Controller,
    This
  );
  gBS->FreePool ((VOID **) UndiPrivateData);
  return Status;
}
```

AI generated projects 登录复制

```
    然后根据UndiPrivateData去:
```

c AI generated projects 登录复制 run

```c
    Status = OpenControllerProtocols (
               Controller,
               This,
               UndiPrivateData
             );
```

AI generated projects 登录复制

```
    再然后就是初始化控制器:
```

c AI generated projects 登录复制 run

```c
Status = InitController (UndiPrivateData);
```

AI generated projects 登录复制

```
    初始化控制器这里面主要是:
    1.获取此控制器支持的PCI命令选项。
    2.设置PCI命令选项以启用设备内存映射IO、端口IO和总线母版。
    3.为传输和接收资源分配内存。
    4.执行第一次硬件初始化
    这里我们重点分析一下第4点:
    Status = GmacFirstTimeInit (&UndiPrivateData->NicInfo);

    在UNDI_PRIVATE_DATA结构体中我们可以看到NicIfo的类型是:
    GIG_DRIVER_DATA                         NicInfo;
    GIG_DRIVER_DATA 的结构如下:
```

收起 ^

c AI generated projects 登录复制 run

```c
typedef struct DRIVER_DATA_S {
  UINT16               State; // stopped, started or initialized
  synopGMACNetworkAdapter  *GmacHw;
  UINTN                Segment;
  UINTN                Bus;
  UINTN                Device;
  UINTN                Function;
  UINT8                PciClass;
  UINT8                PciSubClass;
```

展开 v

AI generated projects 登录复制

```
再看第一次初始化Gmac硬件的代码:
```

c AI generated projects 登录复制 run

```c
/** This function is called as early as possible during driver start
    to ensure the
   hardware has enough time to autonegotiate when the
   real SNP device initialize call is made.
   这个函数在驱动程序启动期间尽可能早地被调用，以确保在真正的SNP设备初始化调用时，
   硬件有足够的时间进行自动协商。
**/
EFI_STATUS
GmacFirstTimeInit (
  GIG_DRIVER_DATA *GigAdapter
  )
```

AI Assistant

```
{
  PCI_CONFIG_HEADER *PciConfigHeader;
  UINT32 *          TempBar;
  UINT8             BarIndex;
  EFI_STATUS        Status;
  UINT32            ScStatus;

  DEBUGPRINT (E1000, ("GmacFirstTimeInit\n"));

  GigAdapter->DriverBusy = FALSE;

  // Read all the registers from the device's PCI Configuration space
  GigAdapter->PciIo->Pci.Read (
                        GigAdapter->PciIo,
                        EfiPciIoWidthUint32,
                        0,
                        MAX_PCI_CONFIG_LEN,
                        GigAdapter->PciConfig
                       );

  PciConfigHeader = (PCI_CONFIG_HEADER *) GigAdapter->PciConfig;

  // Enumerate through the PCI BARs for the device to determine which one is
  // the IO BAR.  Save the index of the BAR into the adapter info structure.
  //通过设备的PCI BAR枚举，以确定哪个是IO BAR。将BAR的索引保存到适配器信息结构中。
  TempBar = &PciConfigHeader->BaseAddressReg0;
  for (BarIndex = 0; BarIndex <= 5; BarIndex++) {
    DEBUGPRINT (E1000, ("BAR = %X\n", *TempBar));
    if ((*TempBar & PCI_BAR_MEM_MASK) == PCI_BAR_MEM_64BIT) {

      // This is a 64-bit memory bar, skip this and the
      // next bar as well.
      TempBar++;
    }

    // Find the IO BAR and save it's number into IoBar
    if ((*TempBar & PCI_BAR_IO_MASK) == PCI_BAR_IO_MODE) {

      // Here is the IO Bar - save it to the Gigabit adapter struct.
      GigAdapter->IoBarIndex = BarIndex;
      break;
    }

    // Advance the pointer to the next bar in PCI config space
    TempBar++;
  }
    GigAdapter->PciIo->GetLocation (
                       GigAdapter->PciIo,
                       &GigAdapter->Segment,
                       &GigAdapter->Bus,
                       &GigAdapter->Device,
                       &GigAdapter->Function
                      );

  DbgPrint (DEBUG_INFO,"  GigAdapter->IoBarIndex = %X\n", GigAdapter->IoBarIndex); //0
  DbgPrint (DEBUG_INFO,"  PCI Command Register   = %X\n", PciConfigHeader->Command);
  //67 根据这里面的数据,结合Command寄存器里面的各个位的含义可知Gmac控制器支持IO请求,响应存储器请求,可以作为主设备等
  DbgPrint (DEBUG_INFO,"  PCI Status Register    = %X\n", PciConfigHeader->Status);//0
  DbgPrint (DEBUG_INFO,"  PCI VendorID           = %X\n", PciConfigHeader->VendorId);//14
  DbgPrint (DEBUG_INFO,"  PCI DeviceID           = %X\n", PciConfigHeader->DeviceId);//7A03
  DbgPrint (DEBUG_INFO,"  PCI SubVendorID        = %X\n", PciConfigHeader->SubVendorId);//0
  DbgPrint (DEBUG_INFO,"  PCI SubSystemID        = %X\n", PciConfigHeader->SubSystemId);//0
  DbgPrint (DEBUG_INFO,"  PCI Segment            = %X\n", GigAdapter->Segment);//0
  DbgPrint (DEBUG_INFO,"  PCI Bus                = %X\n", GigAdapter->Bus);//0
  DbgPrint (DEBUG_INFO,"  PCI Device             = %X\n", GigAdapter->Device);//3
```

```
  DbgPrint (DEBUG_INFO,"  PCI Function           = %X\n", GigAdapter->Function);//0
  ZeroMem (GigAdapter->BroadcastNodeAddress, PXE_MAC_LENGTH);
  SetMem (GigAdapter->BroadcastNodeAddress, PXE_HWADDR_LEN_ETHER, 0xFF);

  GigAdapter->GmacHw->synopGMACdev->MacBase             = (UINTN) (((PciConfigHeader->BaseAddressReg0)& 0xFFFFFFF0) | PCI_REG_BASE);

  GigAdapter->GmacHw->synopGMACdev->DmaBase             = GigAdapter->GmacHw->synopGMACdev->MacBase + DMABASE;

  GigAdapter->GmacHw->synopGMACdev->Version             = synopGMAC_read_version(GigAdapter->GmacHw->synopGMACdev);
  GigAdapter->GmacHw->synopGMACdev->vendor_id           = PciConfigHeader->VendorId;
  GigAdapter->GmacHw->synopGMACdev->device_id           = PciConfigHeader->DeviceId;
  GigAdapter->GmacHw->synopGMACdev->subsystem_vendor_id = PciConfigHeader->SubVendorId;
  GigAdapter->GmacHw->synopGMACdev->subsystem_device_id = PciConfigHeader->SubSystemId;
  GigAdapter->GmacHw->synopGMACdev->revision_id         = (UINT8) PciConfigHeader->RevId;
  GigAdapter->GmacHw->synopGMACdev->function_id         = GigAdapter->Function;
  GigAdapter->GmacHw->synopGMACdev->MaxReinitCnt        = 0;

#define DEBUG_PRINT
#ifdef DEBUG_PRINT
  DbgPrint(DEBUG_INFO,"  GigAdapter->Hw.GmacDev.MacBase:0x%llx\n",GigAdapter->GmacHw->synopGMACdev->MacBase);
  //0x80000e0027278000
  DbgPrint(DEBUG_INFO,"  GigAdapter->Hw.GmacDev.DmaBase:0x%llx\n",GigAdapter->GmacHw->synopGMACdev->DmaBase);
  //0x80000E0027279000
  DbgPrint(DEBUG_INFO,"  subsystem_vendor_id           :0x%08x\n",PciConfigHeader->SubVendorId);//0x00000000
  DbgPrint(DEBUG_INFO,"  subsystem_device_id           :0x%08x\n",PciConfigHeader->SubSystemId);//0x00000000
  DbgPrint(DEBUG_INFO,"  Version                       :0x%08x\n",GigAdapter->GmacHw->synopGMACdev->Version);//0x0000D137
#endif
  GigAdapter->GmacHw->synopGMACdev->mac.autoneg            = TRUE;
  GigAdapter->GmacHw->synopGMACdev->phy.autoneg_wait_to_complete = FALSE;
  GigAdapter->GmacHw->synopGMACdev->phy.reset_disable      = FALSE;
  GigAdapter->GmacHw->synopGMACdev->phy.autoneg_advertised = GMAC_ALL_SPEED_DUPLEX;
  //全双工
  GigAdapter->GmacHw->synopGMACdev->phy.autoneg_mask       = AUTONEG_ADVERTISE_SPEED_DEFAULT;

  GigAdapter->PciClass    = (UINT8) ((PciConfigHeader->ClassId & PCI_CLASS_MASK) >> 8);
  GigAdapter->PciSubClass = (UINT8) (PciConfigHeader->ClassId) & PCI_SUBCLASS_MASK;

  GigAdapter->PciClass    = (UINT8) ((PciConfigHeader->ClassId & PCI_CLASS_MASK) >> 8);
  GigAdapter->PciSubClass = (UINT8) (PciConfigHeader->ClassId) & PCI_SUBCLASS_MASK;

  if (gmac_set_mac_type (GigAdapter->GmacHw->synopGMACdev) != GMAC_SUCCESS) {
    DEBUGPRINT (CRITICAL, ("Unsupported MAC type!\n"));
    return EFI_UNSUPPORTED;
  }

  if (gmac_setup_init_funcs (GigAdapter->GmacHw) != GMAC_SUCCESS) {
    DEBUGPRINT (CRITICAL, ("gmac_setup_init_funcs failed!\n"));
    return EFI_UNSUPPORTED;
  }

  ScStatus = gmac_init_hw (GigAdapter->GmacHw);
  if (ScStatus == GMAC_SUCCESS) {
    Status = EFI_SUCCESS;
    GigAdapter->HwInitialized = TRUE;
  } else {
    GigAdapter->HwInitialized = FALSE;
    Status = EFI_DEVICE_ERROR;
  }

  GigAdapter->CurTxInd = 0;
  GigAdapter->XmitDoneHead = 0;
  GigAdapter->CurRxInd = 0;
```

```
  return Status;
}
```

收起 ^

AI generated projects 登录复制

```
这里面重点说一下: gmac_setup_init_funcs
```

c AI generated projects 登录复制 run

```c
s32 gmac_setup_init_funcs(synopGMACNetworkAdapter *hw)
{
  s32 ret_val = -1;
  synopGMACdevice *Hw;

  ASSERT(hw != NULL);

  Hw = hw->synopGMACdev;

  ret_val = gmac_set_mac_type(Hw);
  if (ret_val) {
    DbgPrint(DEBUG_INFO,"mac type error!\n");
    return ret_val;
  }

  gmac_init_mac_info(Hw);
  gmac_init_phy_info(Hw);
  gmac_init_drv_info(Hw);

  return 0;
}
```

收起 ^

c AI generated projects 登录复制 run

```c
void gmac_init_mac_info(synopGMACdevice *hw)
{
  u32 offset = 0, i = 0;
  u8  function = 0;
  bool InvalidMac = FALSE;
  EFI_LS_SERVICE_PROTOCOL   *Interface = NULL;
  EFI_STATUS  Status;

  gmac_mac_info *mac = &hw->mac;
  function = hw->function_id;

  ASSERT(mac != NULL );

  mac->ops.init       = synopGMAC_mac_init;
  mac->ops.reset      = synopGMAC_reset;
  mac->ops.power_up   = synopGMAC_linux_powerup_mac;
  mac->ops.power_down = synopGMAC_linux_powerdown_mac;

  Status = gBS->LocateProtocol (&gEfiLsServiceProtocolGuid, NULL, (VOID **)&Interface);
  ASSERT_EFI_ERROR (Status);

  offset = function == 0 ? 0x0:0x10;

  Status = Interface->ChipsetSpi.Read((EFI_LS_SPI_PROTOCOL *)&(Interface->ChipsetSpi),EfiDataWidthUint8,ETH_ADDR_LEN,offset,(VOID *)mac->addr);
  Status = Interface->ChipsetSpi.Read((EFI_LS_SPI_PROTOCOL *)&(Interface->ChipsetSpi),EfiDataWidthUint8,ETH_ADDR_LEN,offset,(VOID *)mac->perm_addr);

  CHECK_MACADDR(mac->addr,InvalidMac);


```

```
  if(InvalidMac) {
    DbgPrint(DEBUG_INFO,"  ###Read Mac Addr from 7a spi is invalid, Use Default Mac Addr!\n ");
    if(function == 0 ){
      CopyMem(mac->addr,mac0_addr,ETH_ADDR_LEN);
      CopyMem(mac->perm_addr,mac0_addr,ETH_ADDR_LEN);
    }else if( function == 1){
      CopyMem(mac->addr,mac1_addr,ETH_ADDR_LEN);
      CopyMem(mac->perm_addr,mac1_addr,ETH_ADDR_LEN);
    }else{
      ASSERT(0);
    }
  }

  DbgPrint(DEBUG_INFO,"  Gmac%d MAC ADDR:",function);
  for(i = 0; i< ETH_ADDR_LEN; i++){
    DbgPrint(DEBUG_INFO," 0x%x",mac->addr[i]);
  }
  DbgPrint(DEBUG_INFO,"\n");
}
```

收起 ^

AI generated projects 登录复制

可以看出来通过spi设置了mac地址.

再简要看一下:

c AI generated projects 登录复制 run

```c
void gmac_init_phy_info(synopGMACdevice *hw)
{
  gmac_phy_info *phy = &hw->phy;
  ASSERT( phy != NULL);

  phy->ops.init   = init_phy;
  phy->media_type = gmac_media_type_copper;
}

void gmac_init_drv_info(synopGMACdevice *hw)
{
  gmac_drv_info *drv = &hw->drv;
  ASSERT( drv != NULL);

  drv->attach  = synopGMAC_attach;
  drv->open    = synopGMAC_linux_open;
  drv->close    = synopGMAC_linux_close;
}

```

收起 ^

AI generated projects 登录复制

GmacFirstTimeInit 里面最后一个函数是:
ScStatus = gmac_init_hw (GigAdapter->GmacHw);
它主要完成了硬件的初始化,这里不再细说.
这样GmacFirstTimeInit就执行完了.
InitController函数执行完毕之后,还执行了:

c AI generated projects 登录复制 run

```c
    Status = InitControllerProtocols (
               UndiPrivateData,
               Controller
```

AI Assistant

```
            );
```

AI generated projects 登录复制

InitControllerProtocols 里面安装了gEfiNiiPointerGuid协议
那么Status = InitController (UndiPrivateData);这个函数也就基本上执行完毕了.

start函数里面再往下是:

c AI generated projects 登录复制 run

```c
  if (InitializeChild) {
    DbgPrint (EFI_D_INFO,"ly_test----InitUndiStructures now----\n");
    InitUndiStructures (UndiPrivateData);

    Status = InitChild (UndiPrivateData);
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("InitChild failed with %r\n", Status));
      goto UndiErrorDeleteDevicePath;
    }

    Status = InitChildProtocols (
               UndiPrivateData
             );
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("InitChildProtocols failed with %r\n", Status));
      goto UndiErrorDeleteDevicePath;
    }
    Status = OpenChildProtocols (
               UndiPrivateData,
               This,
               Controller
             );
    if (EFI_ERROR (Status)) {
      DEBUGPRINT (CRITICAL, ("OpenChildProtocols failed with %r\n", Status));
    }
    UndiPrivateData->IsChildInitialized = TRUE;
  }

```

收起 ^

AI generated projects 登录复制

首先看一下 InitUndiStructures (UndiPrivateData);

c AI generated projects 登录复制 run

```c
/** Initializes UNDI (PXE) structures
    初始化PXE的结构

   @param[in]       UndiPrivateData        Private data structure

   @retval          None
**/
VOID
InitUndiStructures (
  IN UNDI_PRIVATE_DATA *UndiPrivateData
  )
{
  // the IfNum index for the current interface will be the total number
  // of interfaces initialized so far
  GigUndiPxeUpdate (&UndiPrivateData->NicInfo, mGMACPxe31);
  InitUndiCallbackFunctions (&UndiPrivateData->NicInfo);

```

AI Assistant

```
}
```

收起

AI generated projects 登录复制

```
在往下是
Status = InitChild (UndiPrivateData); 和
Status = InitChildProtocols (
          UndiPrivateData
        );
 在子句柄中也安装gEfiNiiPointerGuid协议
 这样我们就可以从父句柄或子句柄中获得NII协议。
 注意在InitChildProtocols 里面又调用了
 Status = InitAdapterInformationProtocol (UndiPrivateData);
```

展开

c AI generated projects 登录复制 run

```c
  Status = gBS->CreateEvent (
                 EVT_TIMER | EVT_NOTIFY_SIGNAL,
                 TPL_NOTIFY,
                 GmacReInit,
                 (UndiPrivateData->NicInfo.GmacHw),
                 &(UndiPrivateData->NicInfo.GmacHw->synopGMACdev->ReInit)
                 );

  if (EFI_ERROR (Status)) {
    ASSERT(0);
    return Status;
  }

  Status = gBS->SetTimer (UndiPrivateData->NicInfo.GmacHw->synopGMACdev->ReInit, TimerPeriodic, (UINT64)CHECK_INIT_PERIOD);

```

收起

AI generated projects 登录复制

```
GmacReInit  里面会不断的检查连接模式和链接速度,如果Phy没有连接成功,则会不断的
初始化硬件,初始化Phy并重启mac.
```

c AI generated projects 登录复制 run

```c
u32 gmac_init_hw(synopGMACNetworkAdapter * hw)
{
  s32 ret = 0;
  s32 i = 0;

  ASSERT(hw != NULL);

  for(i = 0; i< RECEIVE_DESC_SIZE; i++){
    if(hw->synopGMACdev->Rxqptr[i] != 0){
      //DbgPrint(DEBUG_INFO,"  FreeRxBuf = 0x%llx\n",hw->synopGMACdev->Rxqptr[i]);
      FreePool((void *)hw->synopGMACdev->Rxqptr[i]);
    }
  }

  DbgPrint(DEBUG_INFO,"  init gmac hardware begin\n");
  if(hw->synopGMACdev->drv.attach != NULL){
    ret = hw->synopGMACdev->drv.attach(hw->synopGMACdev,DEFAULT_PHY_BASE,hw->synopGMACdev->mac.addr);
    if(ret != 0 ){
        DbgPrint(DEBUG_INFO,"### drv.attach Error!\n");
        return ret;
    }
twen
twen
```

```
twen
twen
  25
  26
  27
  28
  29
  30
  31
  32
  33
  34
  35
  36
  37
  38
  39
  40
  41
  42
```

```
    GmacDelay(0x10000);
  }

  if(hw->synopGMACdev->phy.ops.init != NULL){
    ret = hw->synopGMACdev->phy.ops.init(hw->synopGMACdev);
    if(ret != 0 ){
        DbgPrint(DEBUG_INFO,"### phy.ops.init Error!\n");
        return ret;
    }
    GmacDelay(0x10000);
  }

  if(hw->synopGMACdev->mac.ops.reset != NULL){
    ret = hw->synopGMACdev->mac.ops.reset(hw->synopGMACdev);
    if(ret != 0 ){
        DbgPrint(DEBUG_INFO,"### mac.ops.reset Error!\n");
        return ret;
    }
    GmacDelay(0x10000);
  }
```