SMI介绍

软件-亢曼曼

目录

01 SMM和SMI基本知识

02 SMRAM

03 SMI基本流程

04 SMM Driver范例

05 Q&A

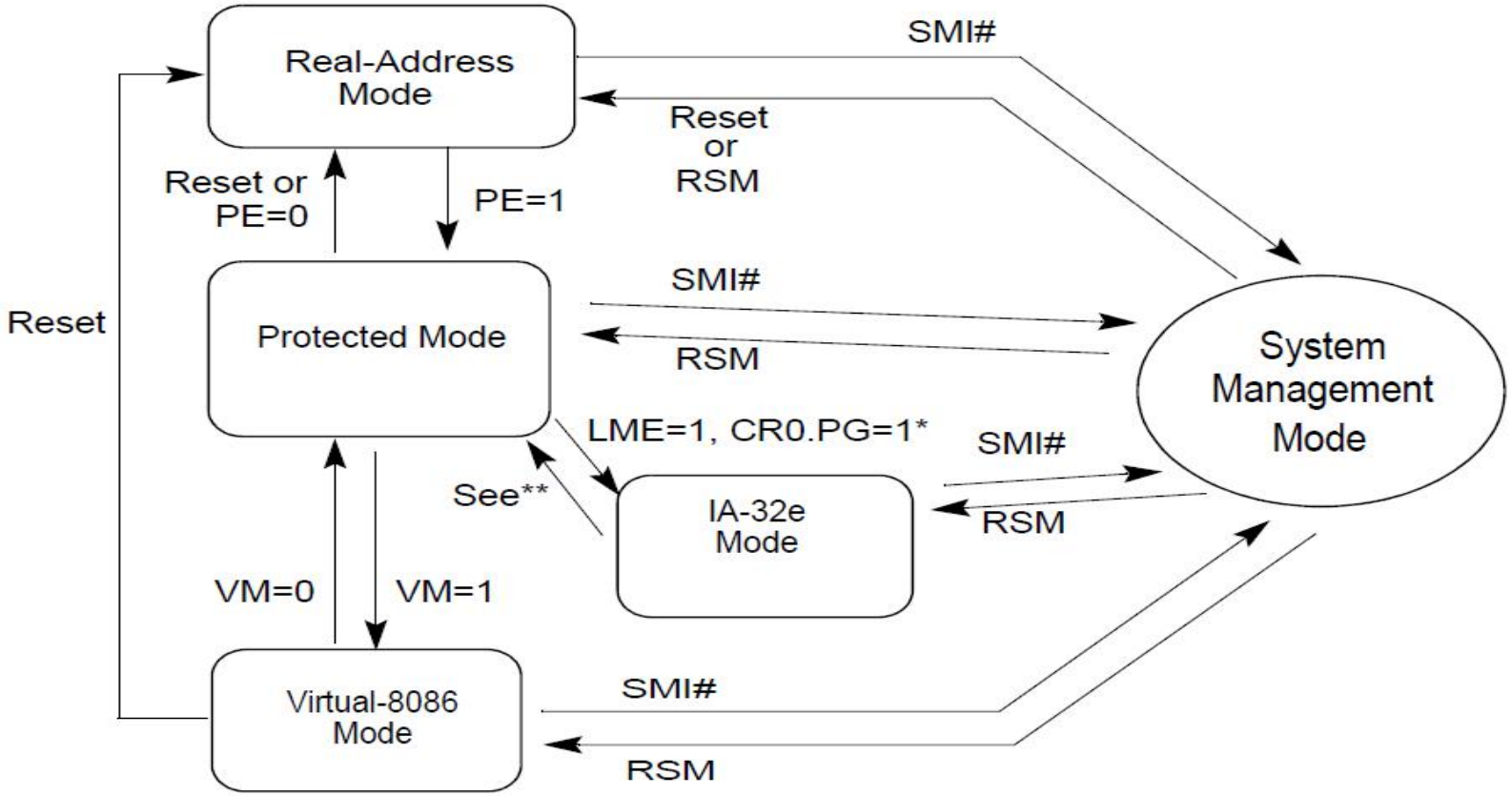
SMM特性

System Management Mode

1. 通过CPU SMI#信号或者APIC消息触发
2. 通过RSM指令退出
3. SMRAM中存储代码&数据
4. 只有BIOS可用，其他任何操作系统或软件不可用
5. 权限最高，可执行任何特权指令和IO操作
6. 默认工作在类似实模式环境，操作数和地址为16位，通过指令前缀可寻址4GB以内地址
7. 一般进入SMM后会切换到保护模式
8. 在SMM中没有地址映射，所有线性地址都对应物理地址
9. 对操作系统透明
10. 可以从CPU的任何模式进入到SMM，退出SMM后返回原本的CPU模式
11. CPU SMI不可屏蔽，进入SMM时会屏蔽其他所有中断
12. SMM不可重入
13. 进入SMM后再产生的下一个SMI会被锁存，退出SMM后响应

SMM模式切换

任何其它模式都能收到SMI中断进入SMM，并在SMM执行完成后返回被中断的模式



SMI

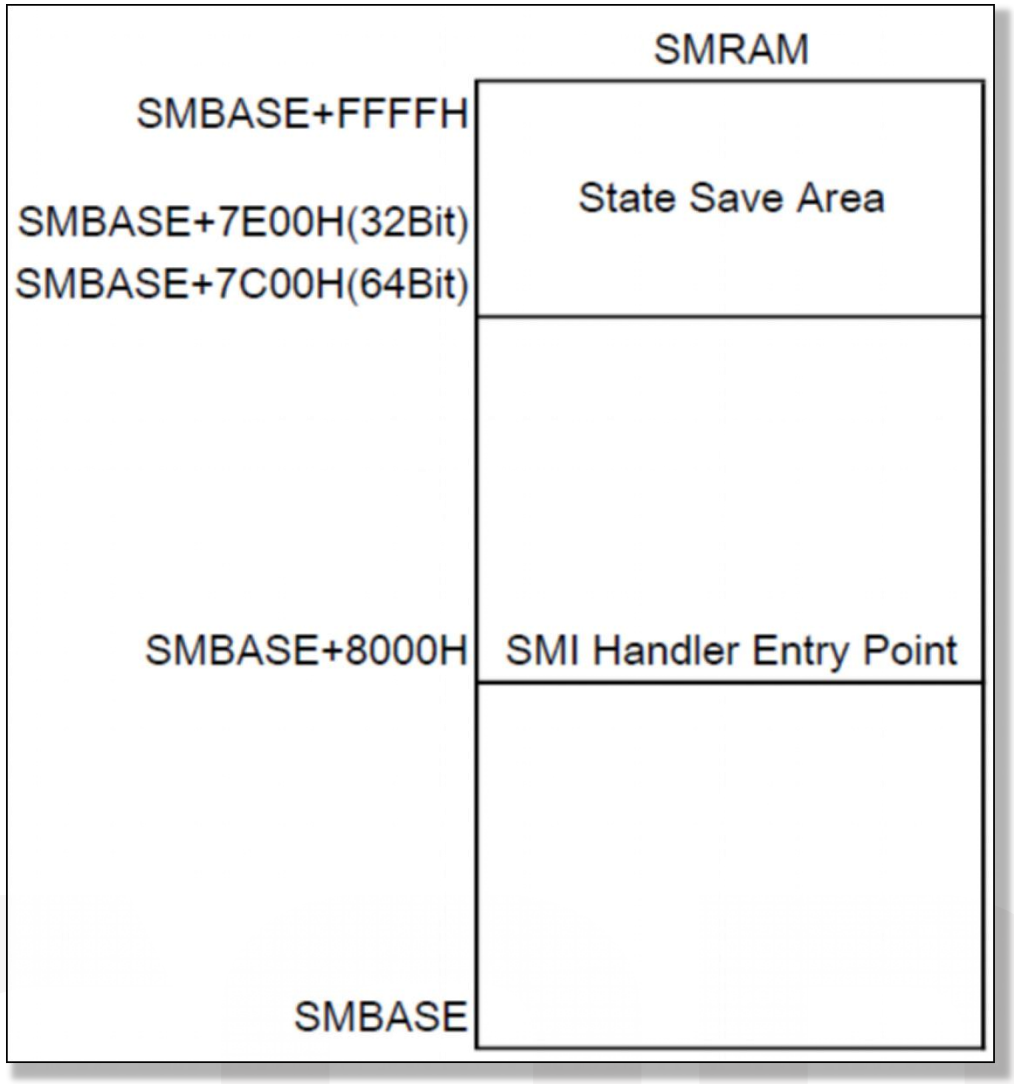
- 进入SMM 的唯一途径 SMI
- SMI是由处理器 SMI#管脚信号有效或者收到 APIC（高级可编程 控制器）总线的SMI消息
- 它是不可屏蔽外部中断 并且独立于其它中断和异常的处理
- SMI优先级高于所有调试中断、NMI、可屏蔽中断和软中断

Priority	Description
1 (Highest)	Hardware Reset and Machine Checks - RESET - Machine Check
2	Trap on Task Switch - T flag in TSS is set
3	External Hardware Interventions - FLUSH - STOPCLK - SMI - INIT
4	Traps on the Previous Instruction - Breakpoints - Debug Trap Exceptions (TF flag set or data/I-O breakpoint)
5	Nonmaskable Interrupts (NMI) [1]
6	Maskable Hardware Interrupts [1]

SMRAM

System Management RAM

- 存放进入SMM之前的系统环境数据
  - CPU进入SMM时将CPU数据存入State Save Area
  - 退出SMM时恢复CPU数据
  - 可通过修改State Save Area数据实现与其他代码/应用的交互
- 存放SMI处理程序、数据、堆栈
- 其他系统数据或OEM数据
- SMRAM在BIOS POST时重定位
  - 默认值为30000H
  - AMD – 可直接修改
  - Intel – 需要进入SMM后修改，退出SMM后生效
  - SMRAM在4GB以内
- BIOS需要初始化每个CPU的SMBASE
- ASeg - SMRAM与VGA Frame Buffer地址共用



SMRAM

➢ SMM_ACCESS2_PROTOCOL

- 主要提供对SMRAM信息的获取和访问控制
- Open()/Close()/Lock()/GetCapabilities()
- LockState/OpenState

```
EFI_SMM_ACCESS2_PROTOCOL gSmmAccess = {
    NBSMM_OpenSmram,
    NBSMM_CloseSmram,
    NBSMM_LockSmram,
    NBSMM_GetCapabilities,
    FALSE,
    FALSE
};
```

SMRAM

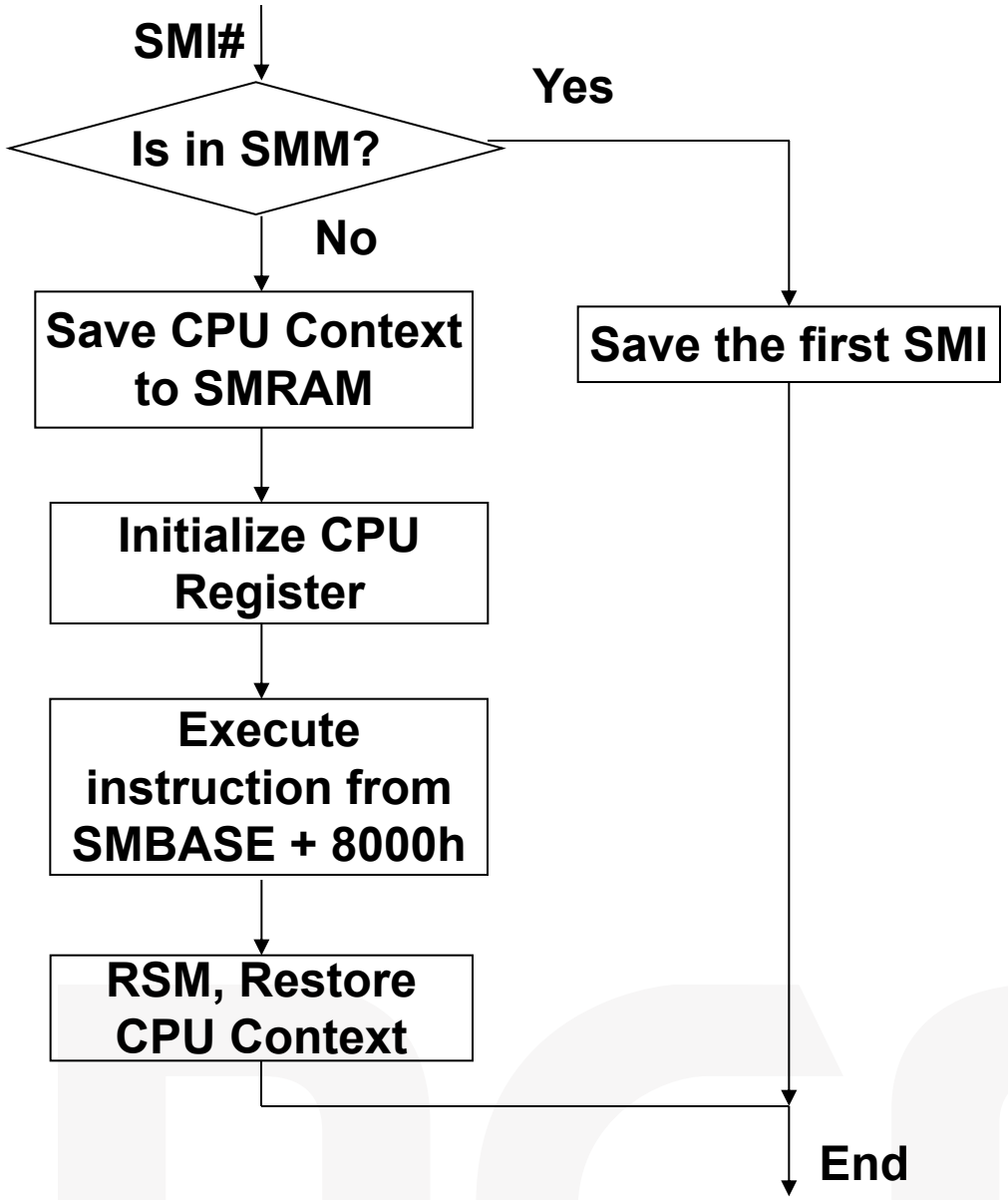
- **TSeg Base**
  - NBSMM_EnableSMMAddress
    - NBSMM_ProgramTsegBase
      - Base = pHIT->EfiMemoryTop
      - Size = PcdTSegSize

  - BSP和所有AP的SMBASE都会被初始化
    - 依赖于MpServices

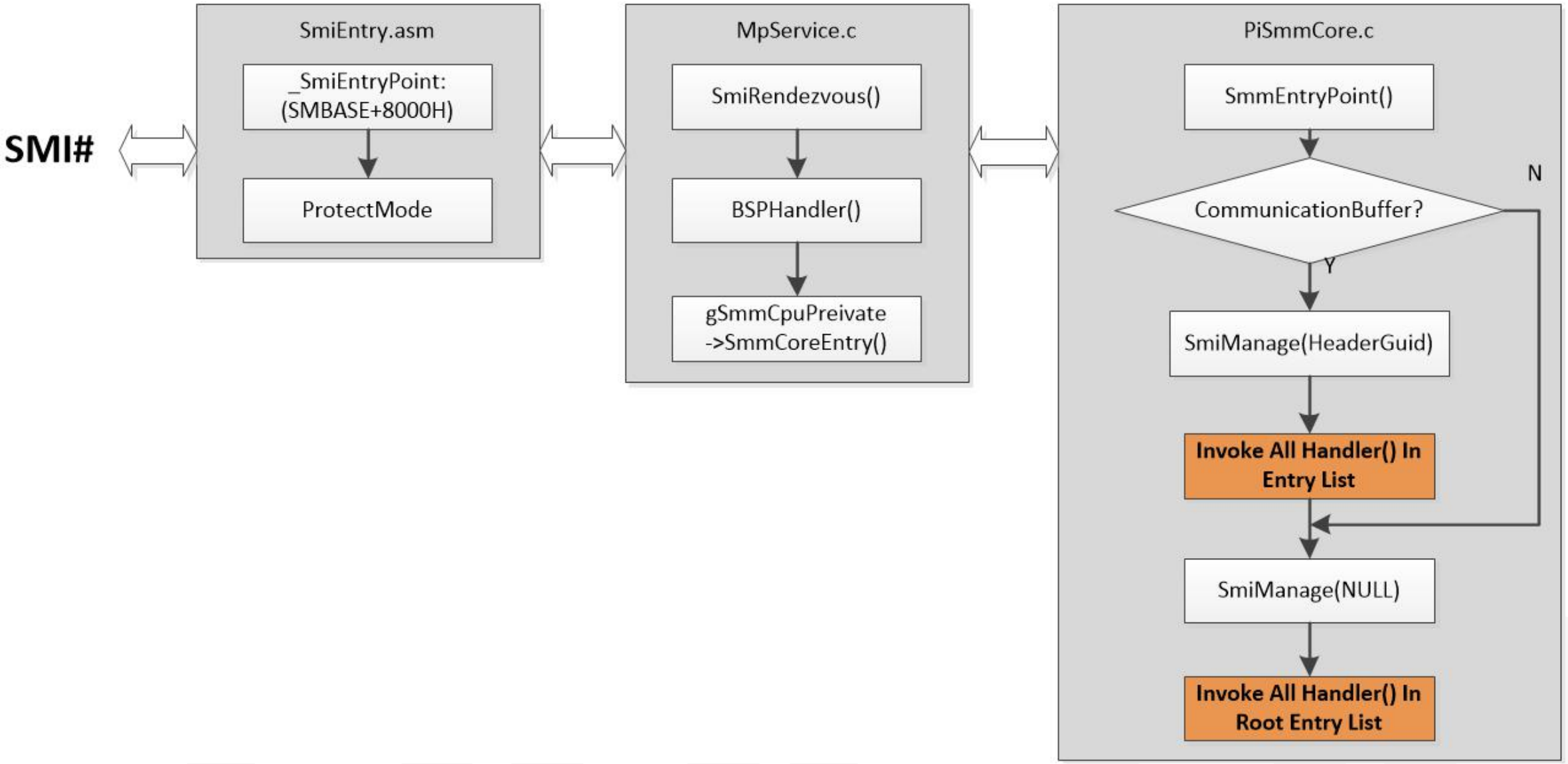
```
NBSMM_ProgramTsegBase ();          // For BSP

// Execute on running APs
Status = gBS->LocateProtocol (&gEfiMpServiceProtocolGuid, NULL, &MpServices);
ASSERT_EFI_ERROR(Status);

MpServices->StartupAllAPs(
      MpServices,                                  // EFI_MP_SERVICES_PROTOCOL*
      (EFI_AP_PROCEDURE)NBSMM_ProgramTsegBase,     // EFI_AP_PROCEDURE
      FALSE,                                       // BOOLEAN SingleThread
      NULL,                                        // EFI_EVENT WaitEvent
      (UINTN)NULL,                                 // UINTN Timeout
      (VOID*)NULL,                                 // VOID *ProcArguments
      NULL);                                       // UINTN *FailedCPUList
```

```
NBSMM_ProgramTsegBase ()
{
  UINT64  qTsegAddress;

  AsmWriteMsr64 (AMD_MSR_SMM_ADDR_HL, gSmramMap[0].PhysicalStart); // TSEG base

  // Program the TSEG size by programming mask register
  qTsegAddress = AsmReadMsr64 (AMD_MSR_SMM_MASK_HL);
  qTsegAddress &= 0x1FFFF;             // Mask off unwanted bits
  qTsegAddress |= (~(UINT64)(gSmramMap[0].PhysicalSize - 1)) & 0xFFFFFFFFFFFE0000;
  qTsegAddress |= 0x00;
  qTsegAddress |= TSEG_CACHING_TYPE;  // Set Tseg cache type as WB=0x6000 or WT=0x4
  AsmWriteMsr64 (AMD_MSR_SMM_MASK_HL, qTsegAddress);

  return EFI_SUCCESS;
}
```

SMI基本流程


SMI#
Is in SMM?
Yes
No
Save CPU Context to SMRAM
Initialize CPU Register
Execute instruction from SMBASE + 8000h
RSM, Restore CPU Context
Save the first SMI
End

SMI Dispatcher

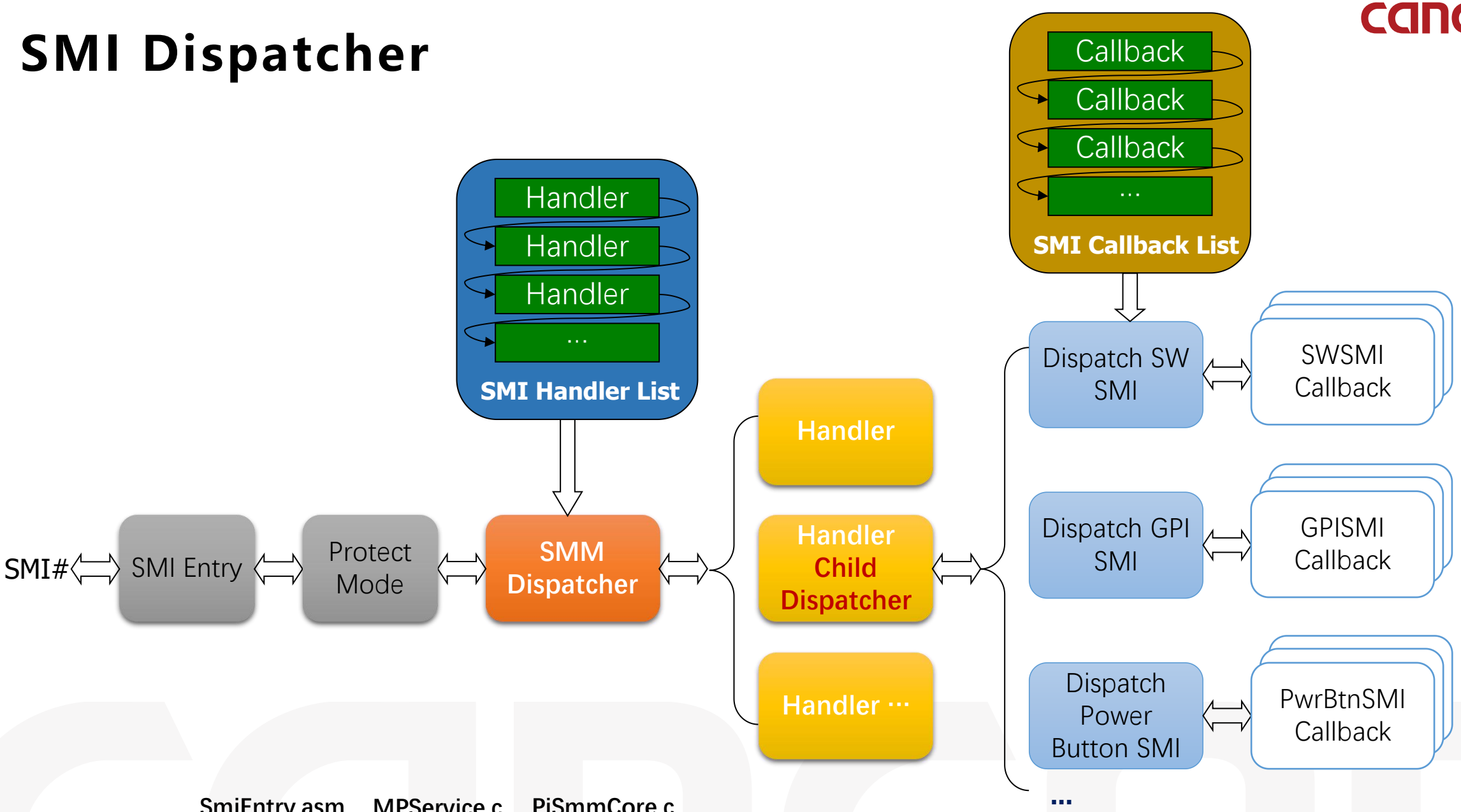


SmiEntry

- 汇编语言代码
- 进入SMM后开始执行的代码
- POST过程中被Copy到SMRAM+8000H位置
- 初始化一些必要的CPU寄存器
- 进入保护模式
- 进入C代码

```
_SmiEntryPoint:
    ;
    ; The encoding of BX in 16-bit addressing mode is the same as of RDI in 64-
    ; bit addressing mode. And that coincidence has been used in the following
    ; "64-bit like" 16-bit code. Be aware that once RDI is referenced as a
    ; base address register, it is actually BX that is referenced.
    ;
    DB      0bbh                        ; mov bx, imm16
    DW      offset _GdtDesc - _SmiEntryPoint + 8000h  ; bx = GdtDesc offset
; fix GDT descriptor
    DB      2eh, 0a1h                   ; mov ax, cs:[offset16]
    DW      DSC_OFFSET + DSC_GDTSIZ
    DB      48h                         ; dec ax
    DB      2eh
    mov     [rdi], eax                  ; mov cs:[bx], ax
    DB      66h, 2eh, 0a1h              ; mov eax, cs:[offset16]
    DW      DSC_OFFSET + DSC_GDTPTR
    DB      2eh
    mov     [rdi + 2], ax               ; mov cs:[bx + 2], eax
    DB      66h, 2eh
    lgdt    fword ptr [rdi]             ; lgdt fword ptr cs:[bx]
; Patch ProtectedMode Segment
    DB      0b8h                        ; mov ax, imm16
    DW      PROTECT_MODE_CS             ; set AX for segment directly
    DB      2eh
    mov     [rdi - 2], eax              ; mov cs:[bx - 2], ax
; Patch ProtectedMode entry
    DB      66h, 0bfh                   ; mov edi, SMBASE
gSmbase    DD    ?
    lea     ax, [edi + (@ProtectedMode - _SmiEntryPoint) + 8000h]
    DB      2eh
    mov     [rdi - 6], ax               ; mov cs:[bx - 6], eax
; Switch into @ProtectedMode
    mov     rbx, cr0
    DB      66h
    and     ebx, 9ffafff3h
    DB      66h
    or      ebx, 00000023h
```

SMI Dispatcher



SMM Driver Example - FchSmmDispatcher

- 提供SMI链表
  - ✓ SwSmi
  - ✓ SleepSmi
  - ✓ PeriodicalSmi
  - ✓ GpiSmi
  - ✓ UsbSmi
  - ✓ Misc
  - ✓ RasSmi
  - ✓ …

```
FCH_SMM_SW_NODE                *HeadFchSmmSwNodePtr;
FCH_SMM_SX_NODE                *HeadFchSmmSxNodePtr;
FCH_SMM_PWRBTN_NODE            *HeadFchSmmPwrBtnNodePtr;
FCH_SMM_PERIODICAL_NODE        *HeadFchSmmPeriodicalNodePtr;
FCH_SMM_PERIODICAL_NODE        *HeadFchSmmPeriodicalNodePtr2;
FCH_SMM_GPI_NODE               *HeadFchSmmGpiNodePtr;
FCH_SMM_USB_NODE               *HeadFchSmmUsbNodePtr;
FCH_SMM_MISC_NODE              *HeadFchSmmMiscNodePtr;
FCH_SMM_APURAS_NODE            *HeadFchSmmApuRasNodePtr;
FCH_SMM_COMMUNICATION_BUFFER *CommunicationBufferPtr;
FCH_SMM_SW_CONTEXT             *EfiSmmSwContext;

EFI_SMM_PERIODIC_TIMER_CONTEXT EfiSmmPeriodicTimerContext;
```

- 初始化所有链表

```
Status = gSmst->SmmAllocatePool (
                  EfiRuntimeServicesData,
                  sizeof (FCH_SMM_SW_NODE),
                  &HeadFchSmmSwNodePtr
                  );
if (EFI_ERROR (Status)) {
  return Status;
}
ZeroMem (HeadFchSmmSwNodePtr, sizeof (FCH_SMM_SW_NODE));
```

SMM Driver Example - FchSmmDispatcher

- **安装Protocol**
  - 提供给其他模块调用**用来注册SMI Callback**
  - Protocol一般都提供Register和Unregister两个Service
  - Register时将SMI Callback记录进链表中
  - 同一个SMI是否可以对应多个SMI Callback由各Dispatch Protocol机制决定，可以是一个或多个

```
FCH_PROTOCOL_LIST FchProtocolList[] = {
  &gFchSmmSwDispatch2ProtocolGuid,           &gFchSmmSwDispatch2Protocol,
  &gEfiSmmSwDispatch2ProtocolGuid,           &gEfiSmmSwDispatch2Protocol,
  &gFchSmmSxDispatch2ProtocolGuid,           &gFchSmmSxDispatch2Protocol,
  &gEfiSmmSxDispatch2ProtocolGuid,           &gEfiSmmSxDispatch2Protocol,
  &gFchSmmPwrBtnDispatch2ProtocolGuid,       &gFchSmmPwrBtnDispatch2Protocol,
  &gEfiSmmPowerButtonDispatch2ProtocolGuid,  &gEfiSmmPwrBtnDispatch2Protocol,
  &gFchSmmPeriodicalDispatch2ProtocolGuid,   &gFchSmmPeriodicalDispatch2Protocol,
  &gEfiSmmPeriodicTimerDispatch2ProtocolGuid,&gEfiSmmPeriodicalDispatch2Protocol,
  &gFchSmmUsbDispatch2ProtocolGuid,          &gFchSmmUsbDispatch2Protocol,
  &gEfiSmmUsbDispatch2ProtocolGuid,          &gEfiSmmUsbDispatch2Protocol,
  &gFchSmmGpiDispatch2ProtocolGuid,          &gFchSmmGpiDispatch2Protocol,
  &gEfiSmmGpiDispatch2ProtocolGuid,          &gEfiSmmGpiDispatch2Protocol,
  &gFchSmmIoTrapDispatch2ProtocolGuid,       &gFchSmmIoTrapDispatch2Protocol,
  &gEfiSmmIoTrapDispatch2ProtocolGuid,       &gEfiSmmIoTrapDispatch2Protocol,
  &gFchSmmMiscDispatchProtocolGuid,          &gFchSmmMiscDispatchProtocol,
                                          };

FCH_PROTOCOL_LIST FchProtocolListRas[] = {
  &gFchSmmApuRasDispatchProtocolGuid,        &gFchSmmApuRasDispatchProtocol,
                                          };
```

```
for (i = 0 ; i < sizeof (FchProtocolList) / sizeof (FCH_PROTOCOL_LIST); i++ ) {
  FchSmmDispatcherHandle =  NULL;
  Status = gSmst->SmmInstallProtocolInterface (
             &FchSmmDispatcherHandle,
             FchProtocolList[i].Guid,
             EFI_NATIVE_INTERFACE,
             FchProtocolList[i].Interface);
  if (EFI_ERROR (Status)) {
    return Status;
  }
}
```

```
FCH_SMM_SW_DISPATCH2_PROTOCOL gFchSmmSwDispatch2Protocol = {
  FchSmmSwDispatch2Register,
  FchSmmSwDispatch2UnRegister,
  (UINTN) MAX_SW_SMI_VALUE
};
```

SMM Driver Example - FchSmmDispatcher

- 注册SMI Handler
- 清除所有SMI状态
- 开放SMI

```
Status = gSmst->SmiHandlerRegister (
                FchSmmDispatchHandler,
                NULL,
                &DispatchHandle
                );
```

```
// Clear all handled SMI status bit
//
  for (SmmDispatcherIndex = 0; SmmDispatcherIndex < NumOfDispatcherTableEntry; SmmDispatcherIndex++ ) {
    SmmDispatcherData32 = ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FchSmmDispatcherTable[SmmDispatcherIndex].StatusReg);
    SmmDispatcherData32 &= FchSmmDispatcherTable[SmmDispatcherIndex].SmiStatusBit;
    ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FchSmmDispatcherTable[SmmDispatcherIndex].StatusReg) = SmmDispatcherData32;
  }
//
// Clear SmiEnB and Set EOS
//
  SmmDispatcherData32 = ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FCH_SMI_REG98);
  SmmDispatcherData32 &= ~(BIT31);
  SmmDispatcherData32 |= BIT28;
  ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FCH_SMI_REG98) = SmmDispatcherData32;
}
```

SMM Driver Example - FchSmmDispatchHandler

1. 遍历SmmDispatcherTable
2. 检测各SMI状态
3. 执行对应SmiDispatcher
4. 开放SMI(Eos bit置1)
5. 检测是否还有SMI需要处理(Eos)
6. 无SMI处理则退出，有则继续此循环

```
ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FCH_SMI_REG98) |= Eos;
EosStatus = ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FCH_SMI_REG98) & Eos;
```

```
for (SmmDispatcherIndex = 0; SmmDispatcherIndex < NumOfDispatcherTableEntry; SmmDispatcherIndex++ ) {
  SmiStatusData = ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FchSmmDispatcherTable[SmmDispatcherIndex].StatusReg);
  if (( SmiStatusData &= FchSmmDispatcherTable[SmmDispatcherIndex].SmiStatusBit) != 0) {
    CommunicationBufferPtr->SmiStatusReg = FchSmmDispatcherTable[SmmDispatcherIndex].StatusReg;
    CommunicationBufferPtr->SmiStatusBit = SmiStatusData;
    CommunicationBuffer = (VOID *) CommunicationBufferPtr;
    Status = FchSmmDispatcherTable[SmmDispatcherIndex].SmiDispatcher (SmmImageHandle, CommunicationBuffer, SourceSize);
    if (Status != EFI_SUCCESS) {
      Status = EFI_WARN_INTERRUPT_SOURCE_PENDING;
    }
    SmmDispatcherIndex = 0;
  }
}
```

SMM Driver Example - FchSmmDispatcherTable

1. 列举各SMI对应的状态寄存器和状态位
2. 提供各SMI对应的处理程序给FchSmmDispatchHandler调用

```
FCH_SMM_DISPATCHER_TABLE   FchSmmDispatcherTable[] = {
  {
    FCH_SMI_REG84,
    UsbSmi,
    FchSmmUsbDispatchHandler
  },
  {
    FCH_SMI_REG88,
    BIT12,
    FchSmmUsbDispatchHandler2
  },
  {
    FCH_SMI_REG88,
    Slp_Type,
    FchSmmSxDispatchHandler
  },
  {
    FCH_SMI_REG88,
    SmiCmdPort,
    FchSmmSwDispatchHandler
  },
  {
    FCH_SMI_REG84,
    PwrBtn,
    FchSmmPwrBtnDispatchHandler
  },
  {
    FCH_SMI_REG90,
    IoTrapping0,
    FchSmmIoTrapDispatchHandler
  },
  {
    FCH_SMI_REG90,
    ShortTimer | LongTimer,
    FchSmmPeriodicalDispatchHandler
  },
  {
    FCH_SMI_REG10,
    Gpe,
    FchSmmGpiDispatchHandler
  },
```

SMM Driver Example - FchSmmSwDispatchHandler

1. 获取当前SwSmi数据
2. 遍历Callback链表
3. 若链表节点匹配，则执行Callback
4. 清除SMI状态

```
do {
  EfiSmmSwContext->SwSmiCpuIndex = i - 1;
  Status = mSmmCpuProtocol->ReadSaveState (
                            mSmmCpuProtocol,
                            sizeof (EFI_SMM_SAVE_STATE_IO_INFO),
                            EFI_SMM_SAVE_STATE_REGISTER_IO,
                            EfiSmmSwContext->SwSmiCpuIndex,
                            &IoInfo
                            );
  if ((Status == EFI_SUCCESS) && (IoInfo.IoPort == SwSmiCmdAddress)
      && ((UINT8) IoInfo.IoData == (UINT8) SwSmiValue)) {
    break;
  }
} while (--i);
EfiSmmSwContext->CommandPort = (UINT8) SwSmiValue;
EfiSmmSwContext->DataPort = (UINT8) (SwSmiValue >> 8);
```

```
while (CurrentFchSmmSwNodePtr->FchSwNodePtr!= NULL) {
  if ((UINT8) CurrentFchSmmSwNodePtr->Context.AmdSwValue == (UINT8) SwSmiValue) {
    if (CurrentFchSmmSwNodePtr->CallBack2Function != NULL) {
      SizeOfFchSmmSwContext = (UINTN) sizeof (FCH_SMM_SW_CONTEXT);
      Status = CurrentFchSmmSwNodePtr->CallBack2Function (
                                       CurrentFchSmmSwNodePtr->DispatchHandle,
                                       &CurrentFchSmmSwNodePtr->Context,
                                       EfiSmmSwContext,
                                       &SizeOfFchSmmSwContext
                                       );
    }
    SwSmiDispatched++;
  }
  CurrentFchSmmSwNodePtr = CurrentFchSmmSwNodePtr->FchSwNodePtr;
}
```

```
  if (SwSmiDispatched <= 0) {
    Status = EFI_NOT_FOUND;
  } else {
    ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FCH_SMI_REG88) = SmiCmdPort;
    Status = EFI_SUCCESS;
  }
}
ACPIMMIO32 (ACPI_MMIO_BASE + SMI_BASE + FCH_SMI_REG88) = SmiCmdPort;
return  Status;
```

SMM Driver Example - SMI Callback Register

- Locate相关SMI Dispatch Protocol
- 初始化Callback相关数据
- 调用Dispatch Protocol提供的Register函数注册Callback

```
Status = gSmst->SmmLocateProtocol (
                  &gEfiSmmSxDispatch2ProtocolGuid,
                  NULL,
                  &SxDispatch
                  );
EntryRegisterContext.Type  = SxS5;
EntryRegisterContext.Phase = SxEntry;
Status = SxDispatch->Register (
                     SxDispatch,
                     RTCWakeupEntryCallBack,
                     &EntryRegisterContext,
                     &S5EntryHandle
                     );
```

```
EFI_STATUS InSmmFunction(
    IN EFI_HANDLE ImageHandle,
    IN EFI_SYSTEM_TABLE *SystemTable )
{
    EFI_STATUS                      Status;
    EFI_HANDLE                      Handle = 0;
    EFI_SMM_SW_DISPATCH2_PROTOCOL   *SwDispatch = NULL;
    EFI_SMM_SW_REGISTER_CONTEXT     SwContext;

    Status = pSmst->SmmLocateProtocol(
        &gEfiSmmSwDispatch2ProtocolGuid,
        NULL,
        (VOID **)&SwDispatch
    );

    if (EFI_ERROR(Status)){
        return Status;
      }
    SwContext.SwSmiInputValue = SW_SMI_OA3_FUNCTION_NUMBER;
    Status = SwDispatch->Register(
        SwDispatch,
        SwSmiOa3UpdateAcpiTable,
        &SwContext,
        &Handle
    );

    return Status;
}
```

DXE Driver触发SMI

- EFI_SMM_CONTROL2_PROTOCOL
  Trigger()

- EFI_SMM_COMMUNICATION_PROTOCOL
  Communicate()

```
Status = gBS->LocateProtocol (
                &gEfiSmmControl2ProtocolGuid,
                NULL,
                (VOID **)&SmmControl
                );

SmiDataValue = PcdGet8 (PcdAmdCcxS3SaveSmi);
SmmControl->Trigger (
                  SmmControl,
                  &SmiDataValue,
                  NULL,
                  0,
                  0
                  );
```

```
Status = gBS->LocateProtocol (&gEfiSmmCommunicationProtocolGuid, NULL,
                              (VOID **) &mSmmCommunication);

SmmCommBufferHeader = (EFI_SMM_COMMUNICATE_HEADER *)mSmmPerformanceBuffer;
SmmPerfCommData = (SMM_PERF_COMMUNICATE *)SmmCommBufferHeader->Data;
ZeroMem((UINT8*)SmmPerfCommData, sizeof(SMM_PERF_COMMUNICATE));
CopyGuid (&SmmCommBufferHeader->HeaderGuid, &gSmmPerformanceProtocolGuid);
SmmCommBufferHeader->MessageLength = sizeof(SMM_PERF_COMMUNICATE);
CommSize = SMM_PERFORMANCE_COMMUNICATION_BUFFER_SIZE;

SmmPerfCommData->Function = SMM_PERF_FUNCTION_GET_GAUGE_ENTRY_NUMBER;
Status = mSmmCommunication->Communicate (mSmmCommunication,
                                         mSmmPerformanceBuffer,
                                         &CommSize);
```

QA

- 注册一个SWSMI，并在SWSMI里读取CMOS 0x46的值

谢谢观看