【UEFI Basics】Timer

原创 jiangwei0512 Posted on 2016-12-11 12:27:08 Read 5.5k Collection 19 Likes 3

Copyright CC 4.0 BY-SA

Category Column: UEFI Development Basics Article Tags: uefi

UEFI Development ... This column includes this content 136 articles Subscribe to our column

Introduction to timers under UEFI

It is mentioned in [UEFI Basics] System Table and Architecture Protocols that the Boot Service Table has the following interfaces:

cpp AI generated projects 登录复制 run

```cpp
//
// Event & Timer Services
//
EFI_CREATE_EVENT                  CreateEvent;
EFI_SET_TIMER                     SetTimer;
EFI_WAIT_FOR_EVENT                WaitForEvent;
EFI_SIGNAL_EVENT                  SignalEvent;
EFI_CLOSE_EVENT                   CloseEvent;
EFI_CHECK_EVENT                   CheckEvent;
```

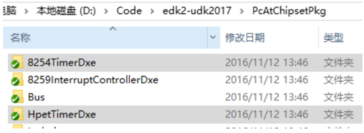
In Architecture Protocols there is one

cpp AI generated projects 登录复制 run

```cpp
EFI_GUID  gEfiTimerArchProtocolGuid = EFI_TIMER_ARCH_PROTOCOL_GUID;
```

These are directly related to the timer.

The timers used in UEFI for Intel platforms are 8254 and HPET:



脑 › 本地磁盘 (D:) › Code › edk2-udk2017 › PcAtChipsetPkg

名称	修改日期	类型
8254TimerDxe	2016/11/12 13:46	文件夹
8259InterruptControllerDxe	2016/11/12 13:46	文件夹
Bus	2016/11/12 13:46	文件夹
HpetTimerDxe	2016/11/12 13:46	文件夹

The newest and most commonly used timer is the HPET timer, which will be the main focus of this article.

Events, polling and other mechanisms under the UEFI architecture all rely on timers.

The timer is initialized in the DXE stage. The end mark of the timer initialization is the installation of the above-mentioned Architectural Protocol:

cpp AI generated projects 登录复制 run

```cpp
//
// Install the Timer Architectural Protocol onto a new handle
//
Status = gBS->InstallMultipleProtocolInterfaces (
                &mTimerHandle,
                &gEfiTimerArchProtocolGuid, &mTimer,
                NULL
                );
ASSERT_EFI_ERROR (Status);
```

Timer initialization

The following mainly talks about the initialization of HPET.

This part is also mentioned in the introduction to the basics of interrupts in the [x86 architecture] . HPET depends on I/O APIC or MSI.

By the way, the 8254 timer depends on the 8259 interrupt controller.

In general, the timer actually relies on the interrupt triggered by the timer.

First of all, the full name of HPET is High Precision Event Timer.

Then introduce the registers corresponding to HPET, as shown in the following table:

Offset Address	register	size	Remark
000h-007h	ID	64-bit	HPET Capabilities and ID
010h-017h	HPET Configure	64-bit	HPET General Configuration Register
020h-027h	HPET Satus	64-bit	Interrupt Status Register
0F0h-0F7h	HPET Main Counter	64-bit	HPET Counter
100h-107h	Timer #0 Configure	64-bit	Timer #0
108h-10Fh	Timer #0 Comparator	64-bit	
120h-127h	Timer #1 Configure	64-bit	Timer #1
128h-12Fh	Timer #1 Comparator	64-bit	
…	There are multiple groups of timers, the middle is omitted		
1E0h-1E7h	Timer #N Configure	64-bit	Timer #N
1E8h-1EFh	Timer #N Comparator	64-bit	

The timer registers here are mainly divided into two categories: global and local. The following is a detailed introduction to these registers:

1. ID register: This register is read-only. The upper 32 bits are the counting frequency of the HPET Main Counter. If the value is xx, it means xxfs (femtoseconds) counts once.

2. Configure register: It has two configuration bits, BIT0 is Overral Enable. Only when it is set to 1, the HPET Main Counter will count; BIT1 is Legacy Replacement Route. When it is set to 1, the use of Timer #0 and Timer #1 is fixed;

3. Status register: records the interrupt status of each Timer;

4. Main Counter register: This is a counter that increases periodically;

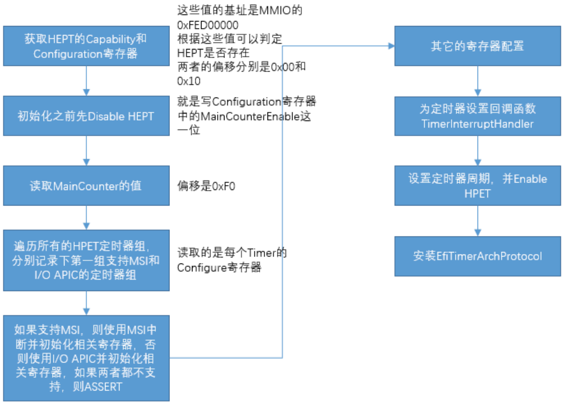
4. Timer#x Comparator register: The "comparison" here refers to the comparison with the Main Counter register. This value needs to be set by ourselves. For example, if we set it to 100, and the Main Counter register increases from 0 to 100, an interrupt will be triggered (then the Comparator value automatically increases to 200, and when the Main Counter increases to 200, an interrupt will be triggered again. The premise here is that this Timer supports periodic triggering);

5. Timer#x Configure register: Regarding the configuration of each Timer, this configuration may not be consistent depending on the Timer.

The above introduction is relatively simple. For a more detailed introduction, you still need to refer to the EDS manual of the corresponding platform.

In addition, the above table does not list the MSI interrupt-related registers of HPET (it should be after the Comparator register of each Timer, but I cannot be sure because there is no manual).

The following is a simple process of HPET initialization:


获取HEPT的Capability和Configuration寄存器
这些值的基址是MMIO的0xFED00000
根据这些值可以判定HEPT是否存在
两者的偏移分别是0x00和0x10
初始化之前先Disable HEPT
就是写Configuration寄存器中的MainCounterEnable这一位
读取MainCounter的值
偏移是0xF0
遍历所有的HPET定时器组，分别记录下第一组支持MSI和I/O APIC的定时器组
读取的是每个Timer的Configure寄存器
如果支持MSI，则使用MSI中断并初始化相关寄存器，否则使用I/O APIC并初始化相关寄存器，如果两者都不支持，则ASSERT
其它的寄存器配置
为定时器设置回调函数TimerInterruptHandler
设置定时器周期，并Enable HPET
安装EfiTimerArchProtocol

For specific code, please refer to PcAtChipsetPkg\HpetTimerDxe\HpetTimer.c in the EDK2 source code.

Timer Application

The application of the timer starts with the timer initialization code:

cpp AI generated projects 登录复制 run

```cpp
    //
    // Initialize I/O APIC entry for HPET Timer Interrupt
    //   Fixed Delivery Mode, Level Triggered, Asserted Low
    //
    IoApicConfigureInterrupt (mTimerIrq, PcdGet8 (PcdHpetLocalApicVector), IO_APIC_DELIVERY_MODE_LOWEST_PRIORITY, TRUE, FALSE);
```

The above code configures the interrupt vector;

The following code sets the interrupt handling function for the interrupt vector:

cpp AI generated projects 登录复制 run

```cpp
  //
  // Install interrupt handler for selected HPET Timer
  //
  Status = mCpu->RegisterInterruptHandler (mCpu, PcdGet8 (PcdHpetLocalApicVector), TimerInterruptHandler);
```

Next, we will study the interrupt handling function TimerInterruptHandler(). In this function, the following code is called periodically:

cpp AI generated projects 登录复制 run

```cpp
    //
    // Call registered notification function passing in the time since the last
    // interrupt in 100 ns units.
    //
    mTimerNotifyFunction (TimerPeriod);
```

The mTimerNotifyFunction here is registered through the TimerDriverRegisterHandler() interface.

This interface is part of EFI_TIMER_ARCH_PROTOCOL:

cpp AI generated projects 登录复制 run

```cpp
///
/// The Timer Architectural Protocol that this driver produces.
///
EFI_TIMER_ARCH_PROTOCOL  mTimer = {
  TimerDriverRegisterHandler,
  TimerDriverSetTimerPeriod,
  TimerDriverGetTimerPeriod,
  TimerDriverGenerateSoftInterrupt
};
```

The TimerDriverRegisterHandler here is EFI_TIMER_ARCH_PROTOCOL.RegisterHandler(), which will be called back after the Architecture Protocol is installed.

The specific execution location is the GenericProtocolNotify() function in DxeProtocolNotify.c. This function will be called after each Architecture Protocol is installed, and for EFI_TIMER_ARCH_PROTOCOL, it corresponds to the following code:

cpp AI generated projects 登录复制 run

```cpp
  //
  // Do special operations for Architectural Protocols
  //

  if (CompareGuid (Entry->ProtocolGuid, &gEfiTimerArchProtocolGuid)) {
    //
    // Register the Core timer tick handler with the Timer AP
    //
    gTimer->RegisterHandler (gTimer, CoreTimerTick);
  }
```

收起 ^

In this case, the CoreTimerTick() function will be called regularly:

cpp AI generated projects 登录复制 run

```cpp
/**
  Called by the platform code to process a tick.

  @param  Duration               The number of 100ns elasped since the last call
                                 to TimerTick

**/
VOID
EFIAPI
CoreTimerTick (
  IN UINT64   Duration
  );
```

收起 ^

In the CoreTimerTick() function, the first event is obtained from mEfiTimerList, and a check is made to see if the event has reached the trigger time. If so, the mEfiCheckTimerEvent event is triggered.

cpp AI generated projects 登录复制 run

```cpp
  //
  // If the head of the list is expired, fire the timer event
  // to process it
  //
  if (!IsListEmpty (&mEfiTimerList)) {
    Event = CR (mEfiTimerList.ForwardLink, IEVENT, Timer.Link, EVENT_SIGNATURE);

    if (Event->Timer.TriggerTime <= mEfiSystemTime) {
      CoreSignalEvent (mEfiCheckTimerEvent);
    }
  }
```

收起 ^

There are a few points to note here:

1. Why only the first one is checked? This is because the event list mEfiTimerList is arranged in chronological order. The first event must be executed first. For details, see the CoreInsertEventTimer() function:

cpp AI generated projects 登录复制 run

```cpp
  //
  // Insert the timer into the timer database in assending sorted order
  //
  for (Link = mEfiTimerList.ForwardLink; Link != &mEfiTimerList; Link = Link->ForwardLink) {
    Event2 = CR (Link, IEVENT, Timer.Link, EVENT_SIGNATURE);

    if (Event2->Timer.TriggerTime > TriggerTime) {
      break;
    }
  }

  InsertTailList (Link, &Event->Timer.Link);
```

收起 ^

2. Here, the obtained event is not directly signaled, but another global event mEfiCheckTimerEvent is signaled. This global event is as follows:

cpp AI generated projects 登录复制 run

```cpp
/**
  Initializes timer support.

**/
VOID
CoreInitializeTimer (
  VOID
  )
{
  EFI_STATUS  Status;

  Status = CoreCreateEventInternal (
             EVT_NOTIFY_SIGNAL,
             TPL_HIGH_LEVEL - 1,
             CoreCheckTimers,
             NULL,
             NULL,
             &mEfiCheckTimerEvent
             );
  ASSERT_EFI_ERROR (Status);
}
```

收起 ^

That is to say, the function CoreCheckTimers() is executed, in which the global event list mEfiTimerList is traversed and events whose time has expired are triggered.

The above is the entire process of triggering events through timers.

Events are closely related to timers, especially timed events.

To add a timer to an event, use the following interface:

```cpp
/**
  Sets the type of timer and the trigger time for a timer event.

  @param[in]  Event             The timer event that is to be signaled at the specified time.
  @param[in]  Type              The type of time that is specified in TriggerTime.
  @param[in]  TriggerTime       The number of 100ns units until the timer expires.
                                A TriggerTime of 0 is legal.
                                If Type is TimerRelative and TriggerTime is 0, then the timer
                                event will be signaled on the next timer tick.
                                If Type is TimerPeriodic and TriggerTime is 0, then the timer
                                event will be signaled on every timer tick.

  @retval EFI_SUCCESS           The event has been set to be signaled at the requested time.
  @retval EFI_INVALID_PARAMETER Event or Type is not valid.
```

```cpp

**/
typedef
EFI_STATUS
(EFIAPI *EFI_SET_TIMER)(
  IN  EFI_EVENT                Event,
  IN  EFI_TIMER_DELAY          Type,
  IN  UINT64                   TriggerTime
  );
```

收起 ︿

It is an interface of gBS and its implementation is as follows:

cpp AI generated projects 登录复制 run

```cpp
/**
  Sets the type of timer and the trigger time for a timer event.

  @param  UserEvent              The timer event that is to be signaled at the
                                 specified time
  @param  Type                   The type of time that is specified in
                                 TriggerTime
  @param  TriggerTime            The number of 100ns units until the timer
                                 expires

  @retval EFI_SUCCESS            The event has been set to be signaled at the
                                 requested time
  @retval EFI_INVALID_PARAMETER  Event or Type is not valid

**/
EFI_STATUS
EFIAPI
CoreSetTimer (
  IN EFI_EVENT            UserEvent,
  IN EFI_TIMER_DELAY      Type,
  IN UINT64               TriggerTime
  )
```

收起 ︿

In this implementation, the event EFI_EVENT is converted to type IEVENT *.

The type of EFI_EVENT is VOID *, and the type of IEVENT is as follows:

cpp AI generated projects 登录复制 run

```cpp
typedef struct {
  UINTN                   Signature;
  UINT32                  Type;
  UINT32                  SignalCount;
  ///
  /// Entry if the event is registered to be signalled
  ///
  LIST_ENTRY              SignalLink;
  ///
  /// Notification information for this event
  ///
  EFI_TPL                 NotifyTpl;
  EFI_EVENT_NOTIFY        NotifyFunction;
  VOID                    *NotifyContext;
  EFI_GUID                EventGroup;
  LIST_ENTRY              NotifyLink;
  UINT8                   ExFlag;
  ///
  /// A list of all runtime events
  ///
  EFI_RUNTIME_EVENT_ENTRY RuntimeData;
  TIMER_EVENT_INFO        Timer;
} IEVENT;
```

收起 ︿

Here we need to pay attention to the type conversion. In fact, the actual type of the event created by gBs->CreateEvent() is IEVENT *, but it is later converted to VOID *.

In the CoreSetTimer() function, the corresponding IEVENT will be loaded into the global variable mEfiTimerList.

In this way, the event is added to the global timer loop event process.

about Us Careers Business Cooperation Seeking coverage ☎ 400-660-0108 ✉ kefu@csdn.net Online Customer Service Working hours 8:30-22:00
Public Security Registration Number 11010502030143 Beijing ICP No. 19004658 Beijing Internet Publishing House [2020] No. 1039-165
Commercial website registration information Beijing Internet Illegal and Harmful Information Reporting Center Parental Control
Online 110 Alarm Service China Internet Reporting Center Chrome Store Download Account Management Specifications
Copyright and Disclaimer Copyright Complaints Publication License Business license
©1999-2025 Beijing Innovation Lezhi Network Technology Co., Ltd.