|  |  |  |
| --- | --- | --- |
| Bochs Developers Guide | | |
| [Prev](http://docs.google.com/logfunctions.html) | Chapter 2. About the code | [Next](http://docs.google.com/cmos-map.html) |

2.6. Internal timers

2.6.1. Overview

The Bochs internal timers are required to provide timer features in the device emulation and for the interaction between simulator and gui. They are implemented in the *bx\_pc\_system\_c* class and driven by the cpu. When programming a timer the interval is specified in useconds and the timer code translates the value to cpu ticks using the [IPS](http://docs.google.com/user/bochsrc.html#BOCHSOPT-CPU-IPS) value. In the original implementation the cpu object calls a timer method to increment the system time by one tick after completing one instruction. If a timer has expired, the related timer handler function is called. Now it is also possible to execute a number of cpu instructions, finally update the timer subsystem with this number and possibly call several timer handlers. Here are some examples for timers in the devices and gui code:

* the PIT (i82C54) system timer at 18.2 Hz
* the CMOS RTC one-second-timer
* the display update timer (set up with "vga: update\_freq=X")
* the devices timer (polls keyboard/mouse events from the gui every 1 emulated msecond)
* the LED auto-off timer (indicating data transfer for min 0.5 seconds)
* the synchronization timers (realtime/slowdown) are also based on the standard timers

These are the capabilities of the Bochs internal timers:

* register / unregister at runtime
* activate / deactivate at runtime
* timer period changeable
* one-shot or continuous mode

2.6.2. Timer definitions, members and methods

Here are the timer-related definitions and members in pc\_system.h:

#define BX\_MAX\_TIMERS 64  
#define BX\_NULL\_TIMER\_HANDLE 10000  
  
typedef void (\*bx\_timer\_handler\_t)(void \*);  
  
 struct {  
 bx\_bool inUse; // Timer slot is in-use (currently registered).  
 Bit64u period; // Timer periodocity in cpu ticks.  
 Bit64u timeToFire; // Time to fire next (in absolute ticks).  
 bx\_bool active; // 0=inactive, 1=active.  
 bx\_bool continuous; // 0=one-shot timer, 1=continuous periodicity.  
 bx\_timer\_handler\_t funct; // A callback function for when the  
 // timer fires.  
 void \*this\_ptr; // The this-> pointer for C++ callbacks  
 // has to be stored as well.  
#define BxMaxTimerIDLen 32  
 char id[BxMaxTimerIDLen]; // String ID of timer.  
 Bit32u param; // Device-specific value assigned to timer (optional)  
 } timer[BX\_MAX\_TIMERS];  
  
 unsigned numTimers; // Number of currently allocated timers.  
 unsigned triggeredTimer; // ID of the actually triggered timer.  
 Bit32u currCountdown; // Current countdown ticks value (decrements to 0).  
 Bit32u currCountdownPeriod; // Length of current countdown period.  
 Bit64u ticksTotal; // Num ticks total since start of emulator execution.  
 Bit64u lastTimeUsec; // Last sequentially read time in usec.  
 Bit64u usecSinceLast; // Number of useconds claimed since then.  
  
 // A special null timer is always inserted in the timer[0] slot. This  
 // make sure that at least one timer is always active, and that the  
 // duration is always less than a maximum 32-bit integer, so a 32-bit  
 // counter can be used for the current countdown.  
 static const Bit64u NullTimerInterval;  
 static void nullTimer(void\* this\_ptr);

These are the public timer-related methods for timer control, driving the timers with the cpu and retrieving the internal time implemented in the *bx\_pc\_system\_c* class:

void initialize(Bit32u ips);  
 int register\_timer(void \*this\_ptr, bx\_timer\_handler\_t, Bit32u useconds,  
 bx\_bool continuous, bx\_bool active, const char \*id);  
 bx\_bool unregisterTimer(unsigned timerID);  
 void setTimerParam(unsigned timerID, Bit32u param);  
 void start\_timers(void);  
 void activate\_timer(unsigned timer\_index, Bit32u useconds, bx\_bool continuous);  
 void deactivate\_timer(unsigned timer\_index);  
 unsigned triggeredTimerID(void) {  
 return triggeredTimer;  
 }  
 Bit32u triggeredTimerParam(void) {  
 return timer[triggeredTimer].param;  
 }  
 static BX\_CPP\_INLINE void tick1(void) {  
 if (--bx\_pc\_system.currCountdown == 0) {  
 bx\_pc\_system.countdownEvent();  
 }  
 }  
 static BX\_CPP\_INLINE void tickn(Bit32u n) {  
 while (n >= bx\_pc\_system.currCountdown) {  
 n -= bx\_pc\_system.currCountdown;  
 bx\_pc\_system.currCountdown = 0;  
 bx\_pc\_system.countdownEvent();  
 // bx\_pc\_system.currCountdown is adjusted to new value by countdownevent().  
 }  
 // 'n' is not (or no longer) >= the countdown size. We can just decrement  
 // the remaining requested ticks and continue.  
 bx\_pc\_system.currCountdown -= n;  
 }  
  
 int register\_timer\_ticks(void\* this\_ptr, bx\_timer\_handler\_t, Bit64u ticks,  
 bx\_bool continuous, bx\_bool active, const char \*id);  
 void activate\_timer\_ticks(unsigned index, Bit64u instructions,  
 bx\_bool continuous);  
 Bit64u time\_usec();  
 Bit64u time\_usec\_sequential();  
 static BX\_CPP\_INLINE Bit64u time\_ticks() {  
 return bx\_pc\_system.ticksTotal +  
 Bit64u(bx\_pc\_system.currCountdownPeriod - bx\_pc\_system.currCountdown);  
 }  
  
 static BX\_CPP\_INLINE Bit32u getNumCpuTicksLeftNextEvent(void) {  
 return bx\_pc\_system.currCountdown;  
 }

This private method is called when the function handling the clock ticks finds that an event has occurred:

void countdownEvent(void);

2.6.3. Detailed functional description

The Bochs timer implementation requires at least one timer to be active. That's why there is a so-called nullTimer to make it work. It is initialized in the constructor on the first timer slot with the highest possible timer interval and it's handler is an empty function.

The most important variables of the timer subsystem are initialized on startup with the nullTimer values and updated after each timer modification (register / unregister / activate / deactivate / processing handler).

* *ticksTotal*: number of ticks total from emulator startup to the last update of timer subsystem
* *currCountdownPeriod*: length of the period from *ticksTotal* to the next timer event
* *currCountdown*: number of ticks remaining until the next timer event occurs

The number if ticks since emulator startup is calculated with the formula *ticksTotal + currCountdownPeriod - currCountdown* and returned with the time\_ticks() method. The number of useconds since emulator startup is returned with the time\_usec() method computed from the return value of time\_ticks() and the [IPS](http://docs.google.com/user/bochsrc.html#BOCHSOPT-CPU-IPS) value.

 To be continued

|  |  |  |
| --- | --- | --- |
| [Prev](http://docs.google.com/logfunctions.html) | [Home](http://docs.google.com/index.html) | [Next](http://docs.google.com/cmos-map.html) |
| Log Functions | [Up](http://docs.google.com/about-the-code.html) | Bochs's CMOS map |