Devices

COLLABORATORS			
	TITLE : Devices		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 23, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Devices

1.1 Amiga® RKM Devices: 13 Timer Device

```
The Amiga timer device provides a general interface to the Amiga's
internal clocks.  Through the timer device, time intervals can be
measured, time delays can be effected, system time can be set and
retrieved, and arithmetic operations can be performed on time values.

                   NEW TIMER FEATURES FOR VERSION 2.0

                Feature                 Description
                --------                -----------
                UNIT_ECLOCK             New timer device unit
                UNIT_WAITUNTIL          New timer device unit
                UNIT_WAITECLOCK         New timer device unit
                ReadEClock()            New function

   Compatibility Warning:
   ----------------------
   The new features for 2.0 are not backwards compatible.

 Timer Device Commands and Functions
 Device Interface
 System Time
 Adding a Time Request
 Using the Time Arithmetic Functions
 E-Clock Time and Its Relationship to Actual Time
 Example Timer Program
 Additional Information on the Timer Device
```

1.2 13 Timer Device / Timer Device Commands and Functions

```
Command         Operation
-------         ---------
TR_ADDREQUEST   Request that the timer device wait a specified period of
                time before replying to the request.

TR_GETSYSTIME   Get system time and place in a timeval structure.
```

TR_SETSYSTIME Set the system time from the value in a timeval structure.

Device Functions

AddTime() Add one timeval structure to another. The result is placed in the first timeval structure.

CmpTime() Compare one timeval structure to another. The result is returned as a longword.

GetSysTime() Get system time and place in a timeval structure.

ReadEClock() Read the current 64 bit value of the E-Clock into an EClockVal structure. The count rate of the E-Clock is also returned. (V36)

SubTime() Subtract one timerequest structure from another. The result is placed in the first timerequest structure.

Exec Functions as Used in This Chapter

AbortIO() Abort a command to the timer device.

CheckIO() Return the status of an I/O request.

CloseDevice() Relinquish use of the timer device. All requests must be complete before closing.

DoIO() Initiate a command and wait for completion (synchronous request).

OpenDevice() Obtain use of the timer device. The timer device may be opened multiple times.

SendIO() Initiate a command and return immediately (asynchronous request).

Exec Support Functions as Used in This Chapter

CreateExtIO() Create an extended I/O request structure of type timerequest. This structure will be used to communicate commands to the timer device.

CreatePort() Create a signal message port for reply messages from the timer device. Exec will signal a task when a message arrives at the reply port.

DeleteExtIO() Delete the timerequest extended I/O request structure created by CreateExtIO().

DeletePort() Delete the message port created by CreatePort().

1.3 13 Timer Device / Device Interface

The timer device operates in a similar manner to the other Amiga devices. To use it, you must first open it, then send I/O requests to it, and then close it when finished. See the Introduction to Amiga System Devices chapter for general information on device usage.

The timer device also provides timer functions in addition to the usual I/O request protocol. These functions still require the device to be opened with the proper timer device unit, but do not require a message port. However, the base address of the timer library must be obtained in order to use the timer functions.

The two modes of timer device operation are not mutually exclusive. You may use them both within the same application.

The I/O request used by the timer device is called timerequest.

```
struct timerequest
{
    struct IORequest tr_node;
    struct timeval tr_time;
};
```

The timer device functions are passed a time structure, either timeval for non E-Clock units or EClockVal for E-Clock units.

```
struct timeval
{
     ULONG tv_secs;   /* seconds */
     ULONG tv_micro;  /* microseconds */
};

struct EClockVal
{
    ULONG ev_hi;   /* Upper longword of E-Clock time */
    ULONG ev_lo;   /* Lower longword of E-Clock time */
};
```

See the include file devices/timer.h for the complete structure definitions. Time requests fall into three categories:

* Time delay - wait a specified period of time. A time delay causes an application to wait a certain length of time. When a time delay is requested, the number of seconds and microseconds to delay are specified in the I/O request.

* Time measure - how long something takes to complete. A time measure is a three-step procedure where the system or E-Clock time is retrieved, an operation or series of operations is performed, and then another time retrieval is done. The difference between the two time values is the measure of the duration of the operation.

* Time alarm - wait till a specific time. A time alarm is a request to be notified when a specific time value has occurred. It is similar to a time delay except that the absolute time value is specified in

the I/O request.

What is an E-Clock?

The E-Clock is the clock used by the Motorola 68000 processor family to communicate with other Motorola 8-bit chips. The E-Clock returns two distinct values - the E-Clock value in the form of two longwords and the count rate (tics/second) of the E-Clock. The count rate is related to the master frequency of the machine and is different between PAL and NTSC machines.

Timer Device Units
Opening The Timer Device
Closing The Timer Device

1.4 13 / Device Interface / Timer Device Units

There are five units in the timer device.

TIMER DEVICE UNITS

Unit	Use
UNIT_MICROHZ	Interval Timing
UNIT_VBLANK	Interval Timing
UNIT_ECLOCK	Interval Timing
UNIT_WAITUNTIL	Time Event Occurrence
UNIT_WAITECLOCK	Time Event Occurrence

* The VBLANK timer unit is very stable and has a granularity comparable to the vertical blanking time. When you make a timing request, such as "signal me in 21 seconds," the reply will come at the next vertical blank after 21 seconds have elapsed. This timer has very low overhead and may be more accurate then the MICROHZ and ECLOCK units for long time periods. Keep in mind that the vertical blanking time varies depending on the display mode.

* The MICROHZ timer unit uses the built-in precision hardware timers to create the timing interval you request. It accepts the same type of command - "signal me in so many seconds and microseconds." The microhertz timer has the advantage of greater resolution than the vertical blank timer, but it may have less accuracy over long periods of time. The microhertz timer also has much more system overhead, which means accuracy is reduced as the system load increases. It is primarily useful for short-burst timing for which critical accuracy is not required.

* The ECLOCK timer unit uses the Amiga E-Clock to measure the time interval you request. This is the most precise time measure available through the timer device.

* The WAITUNTIL timer unit acts as an alarm clock for time requests. It will signal the task when systime is greater than or equal to a specified time value. It has the same granularity as the VBLANK

```
   timer unit.

*  The WAITECLOCK timer unit acts as an alarm clock for time requests.
   It will signal the task when the E-Clock value is greater than or
   equal to a specified time value.  It has the same granularity as the
   ECLOCK timer unit.

Granularity vs. Accuracy.
-------------------------
Granularity is the sampling frequency used to check the timers.
Accuracy is the precision of a measured time interval with respect to
the same time interval in real-time.  We speak only of granularity
because the sampling frequency directly affects how accurate the
timers appear to be.
```

1.5 13 / Device Interface / Opening The Timer Device

```
Three primary steps are required to open the timer device:

*  Create a message port using CreatePort(). Reply messages from the
   device must be directed to a message port.

*  Create an I/O request structure of type timerequest using
   CreateExtIO().

*  Open the timer device with one of the five timer device units. Call
   OpenDevice() passing a pointer to the  timerequest.

struct MsgPort *TimerMP;   /* Message port pointer */
struct timerequest *TimerIO;  /* I/O structure pointer */

  /* Create port for timer device communications */
if (!(TimerMP = CreatePort(0,0)))
    cleanexit(" Error: Can't create port\n",RETURN_FAIL);

  /* Create message block for device IO */
if (!(TimerIO = (struct timerequest *)
                CreateExtIO(TimerMP)(sizeof timerequest)) )
    cleanexit(" Error: Can't create IO request\n",RETURN_FAIL);

  /* Open the timer device with UNIT_MICROHZ */
if (error=OpenDevice(TIMERNAME,UNIT_MICROHZ,TimerIO,0))
    cleanexit(" Error: Can't open Timer.device\n",RETURN_FAIL);

The procedure for applications which only use the timer device functions
is slightly different:

*  Declare the timer device base address variable TimerBase in the
   global data area.

*  Allocate memory for a timerequest structure and a  timeval structure
   using AllocMem().

*  Call OpenDevice(), passing the allocated timerequest structure.
```

```
*  Set the timer device base address variable to point to the timer
   device base.

struct Library *TimerBase;  /* global library pointer */

struct timerequest *TimerIO;
struct timeval *time1;

  /* Allocate memory for timerequest and timeval structures */
TimerIO=(struct timerequest *)AllocMem(sizeof(struct timerequest),
                                MEMF_PUBLIC | MEMF_CLEAR);
time1=(struct timeval *)AllocMem(sizeof(struct timeval),
                                MEMF_PUBLIC | MEMF_CLEAR);
if (!TimerIO | !time1)
    cleanexit(" Error: Can't allocate memory for I/O structures\n",
                        RETURN_FAIL);

if (error=OpenDevice(TIMERNAME,UNIT_MICROHZ,TimerIO,0))
    cleanexit(" Error: Can't open Timer.device\n",RETURN_FAIL);

  /* Set up pointer for timer functions */
TimerBase = (struct Library *)TimerIO->tr_node.io_Device;
```

1.6 13 / Device Interface / Closing The Timer Device

Each OpenDevice() must eventually be matched by a call to CloseDevice().

All I/O requests must be complete before CloseDevice(). If any requests are still pending, abort them with AbortIO().

```
if (!(CheckIO(TimerIO)))
    {
    AbortIO(TimerIO);    /* Ask device to abort any pending requests */
    }
WaitIO(TimerIO);         /* Clean up */
CloseDevice((struct IORequest *)TimerIO);  /* Close Timer device */
```

1.7 13 Timer Device / System Time

The Amiga has a system time feature provided for the convenience of the developer. It is a monotonically increasing time base which should be the same as real time. The timer device provides two commands to use with the system time. In addition, there are utility functions in utility.library which are very useful with system time. See the "Utility Library" chapter of the Amiga ROM Kernel Reference Manual: Libraries for more information.

The command TR_SETSYSTIME sets the system's idea of what time it is. The system starts out at time "zero" so it is safe to set it forward to the "real" time. However, care should be taken when setting the time backwards.

The command TR_GETSYSTIME is used to get the system time. The timer device does not interpret system time to any physical value. By convention, it tells how many seconds have passed since midnight, January 1, 1978. Your program must calculate the time from this value.

The function GetSysTime() can also be used to get the system time. It returns the same value as TR_GETSYSTIME, but uses less overhead.

Whenever someone asks what time it is using TR_GETSYSTIME, the return value of the system time is guaranteed to be unique and unrepeating so that it can be used by applications as a unique identifier.

System time at boot time.

The timer device sets system time to zero at boot time. AmigaDOS will then reset the system time to the value specified on the boot disk. If the AmigaDOS C:SetClock command is given, this also resets system time.

Here is a program that can be used to determine the system time. The command is executed by the timer device and, on return, the caller can find the data in his request block.

Get_Systime.c

1.8 13 Timer Device / Adding a Time Request

Time delays and time alarms are done by opening the timer device with the proper unit and submitting a timerequest to the device with TR_ADDREQUEST set in io_Command and the appropriate values set in tv_secs and tv_micro.

Time delays are used with the UNIT_MICROHZ, UNIT_VBLANK, and UNIT_ECLOCK units. The time specified in a time delay timerequest is a relative measure from the time the request is posted. This means that the tv_secs and tv_micro fields should be set to the amount of delay required.

When the specified amount of time has elapsed, the driver will send the timerequest back via ReplyMsg(). You must fill in the ReplyPort pointer of the timerequest structure if you wish to be signaled. Also, the number of microseconds must be normalized; it should be a value less than one million.

For a minute and a half delay, set 60 in tv_secs and 500,000 in tv_micro.

```
TimerIO->tr_node.io_Command = TR_ADDREQUEST;
TimerIO->tr_time.tv_secs  = 60;        /* Delay a minute */
TimerIO->tr_time.tv_micro = 500000;    /* and a half     */
DoIO(TimerIO);
```

Time alarms are used with the UNIT_WAITUNTIL and UNIT_WAITECLOCK units. The tv_secs and tv_micro fields should be set to the absolute time value of the alarm. For an alarm at 10:30 tonight, the number of seconds from midnight, January 1, 1978 till 10:30 tonight should be set in tv_secs. The timer device will not return until the time is greater than or equal to the absolute time value.

For our purposes, we will set an alarm for three hours from now by getting the current system time and adding three hours of seconds to it.

```
#define SECSPERHOUR (60*60)
struct timeval *systime;

GetSysTime(systime);   /* Get current system time */

TimerIO->tr_node.io_Command = TR_ADDREQUEST;
TimerIO->tr_time.tv_secs  = systime.tv_secs+(SECSPERHOUR*3);
TimerIO->tr_time.tv_micro = systime.tv_micro;
DoIO(TimerIO);
```

```
Time requests with the E-Clock Units.
-------------------------------------
Time requests with the E-Clock units - UNIT_ECLOCK and
UNIT_WAITECLOCK - work the same as the other units except that the
values specified in their I/O requests are compared against the value
of the E-Clock.  See "E-Clock Time and Its Relationship to Actual Time"
below.
```

Remember, you must never reuse a timerequest until the timer device has replied to it. When you submit a timer request, the driver destroys the values you have provided in the timeval structure. This means that you must reinitialize the time specification before reposting a timerequest.

Keep in mind that the timer device provides a general time-delay capability. It can signal you when at least a certain amount of time has passed. The timer device is very accurate under normal system loads, but because the Amiga is a multitasking system, the timer device cannot guarantee that exactly the specified amount of time has elapsed - processing overhead increases as more tasks are run. High-performance applications (such as MIDI time-stamping) may want to take over the 16-bit counters of the CIA B timer resource instead of using the timer device.

```
Problems with small time requests in V1.3 and earlier versions.
---------------------------------------------------------------
You must also take care  to avoid posting a timerequest of less than
2 microseconds with the UNIT_MICROHZ timer device if you are using
V1.3 or earlier versions of the system software.  In V1.3 and earlier
versions of the Amiga system software, sending a timerequest for 0 or
1 microseconds can cause a system crash.  Make sure all your timer
requests are for 2 microseconds or more when you use the UNIT_MICROHZ
timer with those versions.
```

```
 Multiple Timer Requests
```

1.9 13 / Adding a Time Request / Multiple Timer Requests

Multiple requests may be posted to the timer driver. For example, you can make three timer requests in a row:

```
    Signal me in 20 seconds (request 1)
```

```
    Signal me in 30 seconds (request 2)
    Signal me in 10 seconds (request 3)
```

As the timer queues these requests, it changes the time values and sorts the timer requests to service each request at the desired interval, resulting effectively in the following order:

```
    (request 3) in now+10 seconds
    (request 1) 10 seconds after request 3 is satisfied
    (request 2) 10 seconds after request 1 is satisfied
```

If you wish to send out multiple timer requests, you have to create multiple request blocks. You can do this by allocating memory for each timerequest you need and filling in the appropriate fields with command data. Some fields are initialized by the call to the OpenDevice() function. So, for convenience, you may allocate memory for the timerequest you need, call OpenDevice() with one of them, and then copy the initialized fields into all the other timerequest.

It is also permissible to open the timer device multiple times. In some cases this may be easier than opening it once and using multiple requests. When multiple requests are given, SendIO() should be used to transmit each one to the timer.

```
     Multiple_Timers.c
```

If all goes according to plan, TimerIO[1] will finish first, TimerIO[2] will finish next, and TimerIO[0] will finish last.

1.10 13 Timer Device / Using the Time Arithmetic Functions

As indicated above, the time arithmetic functions are accessed in the timer device structure as if they were a routine library. To use them, you create an IORequest block and open the timer. In the IORequest block is a pointer to the device's base address. This address is needed to access each routine as an offset - for example, _LVOAddTime, _LVOSubTime, _LVOCmpTime - from that base address.

```
     Timer_Arithmetic.c
```

```
 Why Use Time Arithmetic?
```

1.11 13 / Using the Time Arithmetic Functions / Why Use Time Arithmetic?

As mentioned earlier in this section, because of the multitasking capability of the Amiga, the timer device can provide timings that are at least as long as the specified amount of time. If you need more precision than this, using the system timer along with the time arithmetic routines can at least, in the long run, let you synchronize your software with this precision timer after a selected period of time.

Say, for example, that you select timer intervals so that you get 161

signals within each 3-minute span. Therefore, the timeval you would have selected would be 180/161, which comes out to 1 second and 118,012 microseconds per interval. Considering the time it takes to set up a call to set the timer and delays due to task-switching (especially if the system is very busy), it is possible that after 161 timing intervals, you may be somewhat beyond the 3-minute time. Here is a method you can use to keep in sync with system time:

1. Begin.

2. Read system time; save it.

3. Perform your loop however many times in your selected interval.

4. Read system time again, and compare it to the old value you saved. (For this example, it will be more or less than 3 minutes as a total time elapsed.)

5. Calculate a new value for the time interval (timeval); that is, one that (if precise) would put you exactly in sync with system time the next time around. Timeval will be a lower value if the loops took too long, and a higher value if the loops didn't take long enough.

6. Repeat the cycle.

Over the long run, then, your average number of operations within a specified period of time can become precisely what you have designed.

You Can't Do 1+1 on E-Clock Values.

The arithmetic functions are not designed to operate on EClockVal
31}s.

1.12 13 Timer Device / E-Clock Time and Its Relationship to Actual Time

Unlike GetSysTime(), the two values returned by ReadEClock() - tics/sec and the EClockVal structure - have no direct relationship to actual time. The tics/sec is the E-Clock count rate, a value which is related to the system master clock. The EClockVal structure is simply the upper longword and lower longword of the E-Clock 64 bit register.

However, when two EClockVal structures are subtracted from each other and divided by the tics/sec (which remains constant), the result does have a relationship to actual time. The value of this calculation is a measure of fractions of a second that passed between the two readings.

```
/* E-Clock Fractions of a second fragment
 *
 * This fragment reads the E-Clock twice and subtracts the two ev_lo values
 *         time2->ev_lo  - time1->ev_lo
 * and divides the result by the E-Clock tics/secs returned by ReadEClock()
 * to get the fractions of a second
 */

struct EClockVal *time1,*time2;
```

```
ULONG E_Freq;
LONG error;
struct timerequest *TimerIO;

TimerIO  = (struct timerequest *)AllocMem(sizeof(struct timerequest ),
           MEMF_CLEAR | MEMF_PUBLIC);

time1 = (struct EClockVal *)AllocMem(sizeof(struct EClockVal ),
         MEMF_CLEAR | MEMF_PUBLIC);

time2 = (struct EClockVal *)AllocMem(sizeof(struct EClockVal ),
         MEMF_CLEAR | MEMF_PUBLIC);

if (!(error = OpenDevice(TIMERNAME,UNIT_ECLOCK,
              (struct IORequest *)TimerIO,0L)) )
    {
    TimerBase = (struct Library *)TimerIO->tr_node.io_Device;
    E_Freq =  ReadEClock((struct EClockVal *) time1); /* Get initial */
                                                      /*   reading   */

       /*  place operation to be measured here */

    E_Freq =  ReadEClock((struct EClockVal *) time2); /* Get 2nd reading */
    printf("\nThe operation took: %f fractions of a second\n",
            (time2->ev_lo-time1->ev_lo)/(double)E_Freq);

    CloseDevice( (struct IORequest *) TimerIO );
    }
```

The Code Takes Some Liberties.

The above fragment only uses the lower longword of the EClockVal structures in calculating the fractions of a second that passed. This was done to simplify the fragment. Naturally, you would have to at least check the values of the upper longwords if not use them to get an accurate measure.

1.13 13 Timer Device / Additional Information on the Timer Device

Additional programming information on the timer device and the utilities library can be found in their include files and Autodocs. All are contained in the Amiga ROM Kernel Reference Manual: Includes and Autodocs.

Timer Device Information	
INCLUDES	devices/timer.h
	devices/timer.i
	utility/date.h
	utility/date.i
AUTODOCS	timer.doc
	utility.doc