QuitWaveDriver

Syntax

Word QuitWaveDriver ()

This function resets the driver. **IMPORTANT**: This must be called before returning to the DOS.

Parameters

None

Return value

None

WaveInAddBuffer

Syntax

Word WaveInAddBuffer (hWaveIn, lpWaveInHdr, wSize)

Sends a buffer to a waveform input device. When the buffer is full, the application is notified.

Parameters

HWaveIn hWaveIn

Specifies a handle to the waveform device which is to receive the buffer.

LpWaveHdr lpWaveInHdr

Specifies a far pointer to a **WaveHdr** structure that identifies the buffer.

Word wSize

Specifies the size of the WaveHdr structure.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid

WaveInClose

Syntax

Word WaveInClose(hWaveIn)

Closes the specified waveform input device.

Parameters

HWaveIn hWaveIn

Specifies a handle to the waveform input device to be closed.
If the function is successful, the handle is no longer valid after this call.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid

WERR_STILLPLAYING

There are still buffers in the queue

Comments

If there are input buffers that have been sent with **WaveInAddBuffer**, and have not been used, the close operation will fail. Call in **WaveInReset** to mark all pending buffers as done.

WaveInGetNumDevs

Syntax

Word WaveInGetNumDevs()

Retrieves the number of waveform input devices present in the system.

Parameters

None

Returns value

Returns the number of waveform input devices in the system.

WaveInOpen

Syntax

Word **WaveInOpen** (lphWaveIn, wDeviceID, lpFormat, dwCallBack, dwCallBackData, dwFlags)

Opens the specified waveform input device for recording.

Parameters

HWaveIn far *lpWaveIn

Specifies a pointer to a HWaveIn handle. This location is filled with a handle identifying the opened waveform input device. Use this handle to identify the device when calling other waveform input functions.

This parameter may be NULL if the WAVE_FORMAT_QUERY flag is specified for the dwFlags.

Word wDeviceID

Identifies the waveform input device that is to be opened.

LpWaveFormat lpFormat

Specifies a far pointer to a WaveFormat data structure that identifies the desired format for recording the waveform data.

int (far * dwCallBack) (**HWaveIn** dev, **LpWaveHdr** block, **DWord** dwCallBackData)

Specifies the address of a callback function. The callback function is called by the driver during recording to process messages related to the progress of the recording.

Specify NULL for this parameter if no callback is desired.

DWord dwCallbackData

Specifies 32 bits of user defined data that is passed to the callback function.

DWord dwFlags

Specifies flags for opening the device.

WAVE_FORMAT_QUERY

If this flag is specified, the device driver will determine if it supports the given format, but will not actually open the device.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_ALLOCATED
Specified resource is already allocated.

WERR_BADDEVICEID
Specified device is out of range.

WERR_BADTRANSFERMODE
Specified transfer mode is unsupported or unavailable.

WERR_STEREOBADCHANNEL
Invalid channel for stereo output (stereo output is only possible on channel 0).

WERR_STEREONEED2FREECHNL
Could not allocate two consecutive channels for stereo output.

WERR_UNSUPPORTEDFORMAT
Attempted to open with an unsupported wave format.
(This error code not currently supported).

Comments

Use **WaveInGetNumDevs** to determine the number of input devices present in the system. The device ID specified by wDeviceID varies from 0 to one less than the specified number of devices present.

The application should make sure that the transfer mode specified in the lpFormat variable is supported by the hardware configuration. The wave driver does NOT validate a DMA or interrupt transfer. This can be done by calling the appropriate functions in the control chip driver.

WaveInReset

Syntax

Word WaveInReset(hWaveIn)

Stops input on a given waveform device and resets the current position to 0. All pending buffers are marked as done.

Parameters

HWaveIn hWaveIn

Specifies a handle to the input device that is to be reset.

Return value

Returns zero if the function is successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

WaveInStart

Syntax

Word WaveInStart(hWaveIn)

Starts input on a given waveform input device.

Parameters

HWaveIn hWaveIn

Specifies a handle to the input device to be started.

Return value

Returns zero if the function is successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

Comments

Buffers are returned to the client when full or when WaveInReset is called (the dwBytesRecorded field in the header will contain the actual length of the data). If there are no buffers available, the data is thrown away without notification to the client and input will continue.

Calling this function when input is already started will have no effect and 0 will be returned.

WaveOutBreakLoop

Syntax

Word WaveOutReset(hWaveOut)

Breaks a loop on a given waveform device and allows playback to continue with the next block in the driver list.

Parameters

HWaveOut hWaveOut

Specifies a handle to the waveform output device to receive the command.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid

Comments

Waveform looping is controlled by the dwLoops and dwFlags fields in the **WaveHdr** structures passed to the device with **WaveOutWrite**. Use the **WHDR_BEGINLOOP** and **WHDR_ENDLOOP** flags in the **WaveHdr** structure to specify the beginning and ending data blocks for looping. To loop on a single block, specify both flags for the same block. Use the dwLoops field in the **WaveHdr** structure for the first block in the loop to specify the number of loops.

Calling this function when nothing is playing or looping will have no effect and 0 will be returned.

WaveOutClose

Syntax

Word WaveOutClose(hWaveOut)

This function closes the specified waveform output device.

Parameters

HWaveOut hWaveOut

Specifies a handle to the waveform output device to be closed. If the function is successful, the handle is no longer valid after the call.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

WERR_STILLPLAYING

There are still buffers in the device queue.

Comments

If the device is still playing a waveform, the close operation will fail. Use **WaveOutReset** to terminate playback before calling **WaveOutClose**.

WaveOutGetNumDevs

Syntax

Word WaveOutGetNumDevs()

Retrieves the number of waveform output devices present in the system.

Parameters

None

Returns value

Returns the number of waveform output devices in the system.

WaveOutGetVolume

Syntax

Word WaveOutGetVolume(hWaveOut, lpdwVolume)

This function queries the current volume setting of a waveform output device.

Parameters

HWaveOut hWaveOut

Identifies the wave output device.

LPDWord lpdwVolume

Specifies a far pointer to a location that will be filled with the current volume setting.

The high-order word contains the left channel volume and the low-order word contains the right channel volume.

If a device does not support volume control on both left and right channels (if the device is opened in mono), only the right channel value is used.

A value of 0xFFFF specifies full volume and a value of 0x0000 is silence.

Return Value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

Comments

Volume control is supported on the left and right channels only if the device was opened specifying 2 in the nChannel field of the **lpWaveFormat** structure of **WaveInOpen**.

WaveOutOpen

Syntax

Word WaveOutOpen (lphWaveOut, wDeviceId, lpFormat, dwCallBack, dwCallBackData, dwFlags)

Opens a specified waveform output device for playback.

Parameters

HWaveOut far *lphWaveOut

Specifies a pointer to an HWAVEOUT handle. This location is filled with a handle identifying the opened waveform output device.
Use the handle to identify the device when calling other wave ouput functions. This parameter may be NULL if WAVE_FORMAT_QUERY is specified in dwFlags.

Word wDeviceID

Identifies the waveform output device that is to be opened.

LpWaveFormat lpFormat

Specifies a pointer to a WaveFormat structure that identifies the format of the waveform that will be sent to the output device.
The WaveFormat structure is also used to specify the "mode" by which the data will be sent to the hardware (WAVE_TRANF_POLLING, WAVE_TRANSF_INTERRUPT, WAVE_TRANSF_ DMA).

int (far * dwCallBack) (**HWaveOut** dev, **LpWaveHdr** block, **DWord** dwCallBackData)

Specifies the address of a callback function. The callback function is called by the driver during playback to process messages related to the progress of the playback.

Specify NULL for this parameter if no callback is desired.

DWord dwCallbackData

Specifies 32 bits of user defined data that is passed to the callback.

DWORD dwFlags

Specifies flags for opening the device.

WAVE_FORMAT_QUERY

If this flag is specified, the device driver will determine if it supports the given format, but will not actually open the device.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_ALLOCATED
Specified resource is already allocated.

WERR_BADDEVICEID
Specified device is out of range.

WERR_BADTRANSFERMODE
Specified transfer mode is unsupported or unavailable.

WERR_STEREOBADCHANNEL
Invalid channel for stereo output (stereo output is only possible on channel 0).

WERR_STEREONEED2FREECHNL
Could not allocate two consecutive channels for stereo output.

WERR_UNSUPPORTEDFORMAT
Attempted to open with an unsupported wave format.
(This error code not currently supported).

Comments

Use **WaveOutGetNumDevs** to determine the number of output devices present in the system. The device ID specified by wDeviceID varies from 0 to one less than the specified number of devices present.

The application should make sure that the transfer mode specified in the **lpFormat** structure is supported by the hardware configuration. The wave driver does NOT validate a DMA or interrupt transfer. This can be made by calling the appropriate functions in the control chip driver. The wave driver uses information stored in the control chip to determine which interrupt and which DMA line it will use.

WaveOutPause

Syntax

Word WaveOutPause(hWaveOut)

Pauses playback on a specified waveform output device. The current playback position is saved. Use **WaveOutRestart** to resume playback from the current playback position.

Parameters

HWaveOut hWaveOut

Specifies a handle to the waveform output device to be paused.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

Comments

Calling this function when output is already paused will have no effect and 0 will be returned.

WaveOutReset

Syntax

Word WaveOutReset(hWaveOut)

Stops playback on a given waveform output device and resets the current position to 0. All pending playback buffers are marked as done.

Parameters

HWaveOut hWaveOut

Specifies a handle to the waveform output device that is to be reset.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

WaveOutRestart

Syntax

Word WaveOutRestart(hWaveOut)

This function restarts a paused waveform output device.

Parameters

HWaveOut hWaveOut

Specifies a handle to the waveform output device that is to be restarted.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

Comments

Calling this function when the output is not paused will have no effect and 0 will be returned.

WaveOutSetLeftRight

Syntax

Word WaveOutSetLeftRight(hWaveOut, leftRight)

Selects which sides the output will be directed to.

Parameters

HWaveOut hWaveOut

Specifies a handle to the waveform output device that is to be restarted.

Word leftRight

Flags specifying the output direction:

WAVE_STEREO_LEFT
WAVE_STEREO_CENTER
WAVE_STEREO_RIGHT

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid

Comments

This function is useful only when the channel is monophonic. Stereophonic channels are always output left and right.

WaveOutSetVolume

Syntax

Word WaveOutSetVolume(hWaveOut, dwVolume)

Sets the volume of a waveform output device.

Parameters

HWaveOut hWaveOut

Identifies the wave output device.

Dword dwVolume

Specifies the volume setting.

The high-order word contains the left channel volume and the low-order word contains the right channel volume.

If a device does not support volume control on both left and right channels (if the device is opened in mono), only the right channel value is used.

A value of 0xFFFF specifies full volume and a value of 0x0000 is silence.

Return value

Returns zero if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

Comments

Volume control is supported on the left and right channels only if the device was opened specifying 2 in the nChannel field of the **lpWaveFormat** structure specified in **WaveOutOpen**.

Note that this controls output volume only.

WaveOutWrite

Syntax

Word WaveOutWrite(hWaveOut, lpWaveOutHdr, wSize)

Sends a data block to the specified waveform output device.

Parameters

HWaveOut hWaveOut

Specifies a handle to the waveform device that the data is to be sent to.

LpWaveHdr lpWaveOutHdr

Specifies a far pointer to a **WaveHdr** structure containing information about the data block.

Word wSize

Specifies the size of the **WaveHdr** structure.

Return value

Returns 0 if the function was successful. Otherwise, it returns an error code. Possible error codes are:

WERR_INVALIDHANDLE

Specified device handle is invalid.

Comments

Unless playback is paused by **WaveoutPause**, playback begins when the first data block is sent to the device.

When writing to a device opened using the WAVE_TRANSF_POLLING mode, control will be returned to the application only when the buffer has been completely played. Using this transfer mode, wave output must be paused with **WaveOutPause** prior to calling **WaveOutWrite** if the application must write more than one buffer.

5.5 DOS Timer Driver

The Ad Lib Gold card offers to developers 5 multi-purpose timers. They are physically located on two different chips but their implementation are similar.

All timers have their own base clock (time resolution) and counter size (maximum period). The controls available for all timers are:

- Write access in their register of different count values (divider).
- Stop and start (decrementing the initial stored count until it reach zero and re-writing the original count, again and again).
- Enable/disable interrupts to occur on zero count crossing.
- Read the interrupt status (access on the zero count crossing).

Some differences exist and need to be noticed:

- The timer 2 from the MMA chip is the only timer whose current count can be read.
- Yamaha in its own documentation use the terms timer 1 and 2 for the timers physically located in the OPL3 chip and timers located in the MMA chip.
- A base counter (another timer) is used in the MMA chip as an input clock for the timers 1 and 2. Those last two timers are decremented each time the base counter reaches zero. This means that the software must initialized the base counter with an appropriate value then the timer 1 or 2.

Here is a table that illustrates the specifications of all timers:

	OPL3 chip		MMA chip			
	Tim. 1	Tim. 2	Tim. 0	B. C.	Tim. 1	Tim. 2
time resolution in µsec	80	320	1.89	1.89	1.89	1.89
max period length in msec	20.4	81.6	123.83	7.738	116.07	507116
counter size in bits	8	8	16	12	4+12	16+12

Table 1: Hardware specifications of timers

Remember that the MMA timer 1 and 2 are combined with the MMA base counter and that their combined specifications gives for the timer 1 a size of 16 bits and for the timer 2 a size of 28 bits.

The timer's function can be access directly or by the TimerDrvService functions which is a dispatcher.

Each timer function is presented in the following pages.

LoadStartOPL3Timer1

LoadStartOPL3Timer2
LoadStartMMATimer0
LoadStartMMATimer1
LoadStartMMATimer2

Syntax

WORD **LoadStartOPL3Timer1**(void)
WORD **LoadStartOPL3Timer2**(void)
WORD **LoadStartMMATimer0**(void)
WORD **LoadStartMMATimer1**(void)
WORD **LoadStartMMATimer2**(void)

This will load the physical counter with the count associated and start the counter.

Parameters

None

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured when loading.

Comments

None

StopOPL3Timer1

StopOPL3Timer2
StopMMATimer0
StopMMATimer1
StopMMATimer2

Syntax

WORD **StopOPL3Timer1**(void)
WORD **StopOPL3Timer2**(void)
WORD **StopMMATimer0**(void)
WORD **StopMMATimer1**(void)
WORD **StopMMATimer2**(void)

Stop the associated timer.

Parameters

None

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured when stoping.

Comments

None

SetOPL3Timer1Counter

SetOPL3Timer2Counter
SetMMATimer0Counter
SetMMATimer1Counter
SetMMATimer2Counter
SetMMABaseCounterCounter

Syntax

WORD **SetOPL3Timer1Counter**(BYTE count)
WORD **SetOPL3Timer2Counter**(BYTE count)
WORD **SetMMATimer0Counter**(WORD count)
WORD **SetMMATimer1Counter**(BYTE count)
WORD **SetMMATimer2Counter**(WORD count)
WORD **SetMMABaseCounterCounter**(WORD count)

Set the OPL3 and MMA timer with the count value. Base clock periods are the following:

OPL3Timer1:	79.9682 us
OPL3Timer2:	319.873 us
MMATimer0:	1.89 us
MMATimer1:	1.89 us
MMATimer2:	1.89 us
MMATimerBaseCounter:	1.89 us

See table xx for more information the capacity of each timer.

Parameters

BYTE count
WORD count
The parameters count specified the number of cycle the timer is supposed to do. Depending of timer count is BYTE or WORD parameter.

Return value

TIMER_NO_ERROR
If the function was successful.

TIMER_FUNCTION_ERROR
If a problem occured when setting.

Comments

It is important to check the table xx because each timer don't use all of the bits in the count parameters.

SetOPL3Timer1Period

SetOPL3Timer2Period
SetMMATimer0Period
SetMMATimer1Period
SetMMATimer2Period
SetMMABaseCounterPeriod

Syntax

WORD **SetOPL3Timer1Period**(DWORD lPeriod)
WORD **SetOPL3Timer2Period**(DWORD lPeriod)
WORD **SetMMATimer0Period**(DWORD lPeriod)
WORD **SetMMATimer1Period**(DWORD lPeriod)
WORD **SetMMATimer2Period**(DWORD lPeriod)
WORD **SetMMABaseCounterPeriod**(DWORD lPeriod)

This set of functions offer another way to set the count of a timer. The period of a cycle is passed instead of passing the divider. It becomes more easy for the programmer to think in terms of period rather than in terms of a divider to associate with the required period.

Parameters

DWORD lPeriod

Period in usec to be passed to the timer.

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured when setting.

Comments

Check the table xx to be sure to respect the maximum capacity of the timer. The period will be round to the precision of the timer.

EnableOPL3Timer1

EnableOPL3Timer2
EnableMMATimer0
EnableMMATimer1
EnableMMATimer2

Syntax

WORD **EnableOPL3Timer1**(void)
WORD **EnableOPL3Timer2**(void)
WORD **EnableMMATimer0**(void)
WORD **EnableMMATimer1**(void)
WORD **EnableMMATimer2**(void)

This will set the mask bit associated with the timer interrupt.

Parameters

None

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured when enabling.

Comments

None

DisableOPL3Timer1

DisableOPL3Timer2
DisableMMATimer0
DisableMMATimer1
DisableMMATimer2

Syntax

WORD **DisableOPL3Timer1**(void)
WORD **DisableOPL3Timer2**(void)
WORD **DisableMMATimer0**(void)
WORD **DisableMMATimer1**(void)
WORD **DisableMMATimer2**(void)

This will reset the mask bit associated with the timer interrupt.

Parameters

None

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured when disabling.

Comments

None

GetOPL3TimerIntStatus

GetMMATimerIntStatus

Syntax

WORD **GetOPL3TimerIntStatus**(void)
WORD **GetMMATimerIntStatus**(void)

These functions will return the state of timer interrupt of the OPL3 and MMA.

Parameters

None

Return value

OPL3

return 0 if no timer has interrupted.
return 2 if timer 1 has interrupted.
return 1 if timer 2 has interrupted.
return 3 if timer 1 and 2 has interrupted.MMA
return 0 if no timer has interrupted.
return 1 if timer 0 has interrupted.
return 2 if timer 1 has interrupted.
return 4 if tmer 2 has interrupted.
or any combination of 1,2 and 4 if multiple timer has interrupted.

Comments

The MMA chip has a special behavior: it will reset the interrupt bit after a status register reading. Note that this routine is automatically called by the main interrupt handler from the Control Chip Driver. Using GetOPL3TimerIntStatus will not reset the OPL3 status register bits.

AssignOPL3Timer1IntService

AssignOPL3Timer2IntService
AssignMMATimer0IntService
AssignMMATimer1IntService
AssignMMATimer2IntService

Syntax

WORD **AssignOPL3Timer1IntService**(void (*function)(void))
WORD **AssignOPL3Timer2IntService**(void (*function)(void))
WORD **AssignMMATimer0IntService**(void (*function)(void))
WORD **AssignMMATimer1IntService**(void (*function)(void))
WORD **AssignMMATimer2IntService**(void (*function)(void))

Use by applications to assign their callback function on a specific interrupt.

Parameters

void (*function)(void)

The parameter is the callback prototype.

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured with the assign procedure.

Comments

The application user must specifie a callback routine that will automatically be called when the interrupt occurs. This callback function must be very short to execute because this is a timer interrupt that may occurs at a very high rate. At initialisation the default service hooked on each timer interrupt is a local DoNothing function that must be replaced by the application user.

RestoreOPL3Timer1IntService

RestoreOPL3Timer2IntService
RestoreMMATimer0IntService
RestoreMMATimer1IntService
RestoreMMATimer2IntService

Syntax

WORD **RestoreOPL3Timer1IntService**(void)
WORD **RestoreOPL3Timer2IntService**(void)
WORD **RestoreMMATimer0IntService**(void)
WORD **RestoreMMATimer1IntService**(void)
WORD **RestoreMMATimer2IntService**(void)

Use by applications to remove their callback function from the interrupt process.

Parameters

None

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured with the restore procedure.

Comments

None

ExecOPL3Timer1IntService

ExecOPL3Timer2IntService
ExecMMATimer0IntService
ExecMMATimer1IntService
ExecMMATimer2IntService

Syntax

void **ExecOPL3Timer1IntService**(void)
void **ExecOPL3Timer2IntService**(void)
void **ExecMMATimer0IntService**(void)
void **ExecMMATimer1IntService**(void)
void **ExecMMATimer2IntService**(void)

Those routines will execute the function associated with each interrupt.

Parameters

None

Return value

None

Comments

None

ResetOPL3LastTimerInt

Syntax

WORD **ResetOPL3LastTimerInt**(void)

This will reset the IRQ signal generated by timers 1 and 2.

Parameters

None

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured with the reset procedure.

Comments

This function does not exist for the MMA because the MMA clear the status after each reading of the status register.

AllocateOPL3Timer1

AllocateOPL3Timer2
AllocateMMATimer0
AllocateMMATimer1
AllocateMMATimer2
AllocateMMABaseCounter

Syntax

WORD **AllocateOPL3Timer1**(void)
WORD **AllocateOPL3Timer2**(void)
WORD **AllocateMMATimer0**(void)
WORD **AllocateMMATimer1**(void)
WORD **AllocateMMATimer2**(void)
WORD **AllocateMMABaseCounter**(void)

This procedure will reserve and from then denied any external application access to this timer.

Parameters

None

Return value

1: if available
0: if not available

Comments

Any application who wants to use the service of any timers should ask the Timer Driver for its disponibility using an allocation routine. The application should free the timer after use.

FreeOPL3Timer1

FreeOPL3Timer2
FreeMMATimer0
FreeMMATimer1
FreeMMATimer2
FreeMMABaseCounter

Syntax

WORD **FreeOPL3Timer1**(void)
WORD **FreeOPL3Timer2**(void)
WORD **FreeMMATimer0**(void)
WORD **FreeMMATimer1**(void)
WORD **FreeMMATimer2**(void)
WORD **FreeMMABaseCounter**(void)

Free the the timer.

Parameters

None

Return value

1: if operation succed
0: if operation not succed

Comments

None

GetMMATimer2Content

Syntax

WORD **GetMMATimer2Content**(void)

This routine returns the content of the MMA timer 2.

Parameters

None

Return value

16 bit content of MMA timer 2

Comments

This is the only timer that can be read. These timers respect the specification of Windows Multi-Media.

GetOPL3Timer1Caps

GetOPL3Timer2Caps
GetMMATimer0Caps
GetMMATimer1Caps
GetMMATimer2Caps

Syntax

WORD **GetOPL3Timer1Caps**
(DWORD far *lPeriodMin, DWORD far *lPeriodMax)
WORD **GetOPL3Timer2Caps**
(DWORD far *lPeriodMin, DWORD far *lPeriodMax)
WORD **GetMMATimer0Caps**
(DWORD far *lPeriodMin, DWORD far *lPeriodMax)
WORD **GetMMATimer1Caps**
(DWORD far *lPeriodMin, DWORD far *lPeriodMax)
WORD **GetMMATimer2Caps**
(DWORD far *lPeriodMin, DWORD far *lPeriodMax)

Used by external modules to query the driver on physical limits of each timer. It returns the minimum and maximum period covered by the timer in micro seconds.

Parameters

DWORD far *lPeriodMin
DWORD far *lPeriodMax

These two address will receve the minimum and the maximum period capacity respectively of the timer.

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured with the procedure.

Comments

None

InitTimerDriver

Syntax

WORD **InitTimerDriver**(WORD base)

This procedure initialize the Timer Driver structure with default values. This procedure should be used the first time the driver is called.

Parameters

WORD base

Actual address of the Ad Lib control chip.

Return value

TIMER_NO_ERROR

If the function was sucessful.

TIMER_FUNCTION_ERROR

If a problem occured with the procedure.

Comments

None

TimerDrvService

Syntax

WORD far TimerDrvService(WORD segm, WORD offs)

Entry point for the AdLib timer dispatcher. The segment and offset of the argument structure are passed as argument.

Parameters

WORD segm

WORD offs

These two parameters specify the segment and the offset of the following structure which is used to pass parameters to the TimerDrvService routine.

```
struct TimerArgum {
    WORD        controlID;                  which service to be used
    WORD        timerDv;                    on which timer
    DWORD       param;                      optionnal based on service used
    DWORD       param2;                     optionnal based on service used
    void        (interrupt far *function)(); optionnal based on service used
    }
```

Return value

Service result if any.

Comments

See TimerDrv.h for all ID of services.

Chapter 7 - Low-level Programming

7.1 Mixer and Setup Features 1

Register Access 1

Status Register 2
Register Map 2

Register Reference 4

Control/ID 4
Telephone Control 5
Sampling Gain 5
Final Output Volume 5
Bass 6
Treble 7
Output Mode 7
Mixing Volumes 8
Audio Selection 8
Register 12h 9
Audio IRQ/DMA Select - Channel 0 9
DMA Select - Channel 1 10
Audio Relocalisation 11
SCSI IRQ/DMA Select 12
SCSI Relocalization 13
Surround 13

7.2 FM Synthesis 15

Programming the YM3812 17

The Ad Lib Music Synthesizer Card 17
Operators 18
ALMSC Input / Output Map 19
Register Reference 21

Test Register/WSE 21
Timers 21
Status Register 22
CSM/Keyboard Split 23
AM/VIB/EG-TYP/KSR/Multiple 23
KSL/Total Level 25
ADSR 26
BLOCK/F-Number 26
Rhythm/AM Dep/VIB Dep 27
FeedBack/Connection 27
Wave Select 28

Programming the YMF262 29

Register Array 0 29
Register Array 1 33
4-Operator Voices 33

7.3 Digital Input and Output (Digital Audio and MIDI) 37

Register Reference 40

Status Register 40
Register 00H: Test Register 40
Registers 02H - 07H: Timer Counters 41
Register 08H: Timer Control 42

Stand-by Mode 42
Timer Interrupt Masks 42
Timer Controls 42

Register 09H: Playback and Recording Control 42

Reset PCM/ADPCM 42
Select Output Channel 42
Select Frequency 43
PCM/ADPCM Selection 43
Select Record/Playback 43
Start/Stop Record/Playback 43

Register 0AH: Output Volume Control 43
Register 0BH: PCM/ADPCM Data 44
Register 0CH: Sampling Format and Control 44

Interleaving 44
Set Data Format 44
Set FIFO Interrupt 45
FIFO Interrupt Mask 45
DMA Mode Specification 46

Register 0DH: MIDI and Interrupt Control 46

Mask Digital Overrun Error 46
Mask MIDI Overrun Error 46
Reset MIDI transmit circuit 46
Mask MIDI transmit FIFO interrupts 46
Reset MIDI Receive Circuit 46
Mask MIDI Receive FIFO Interrupts 46

Register 0EH: MIDI Data 46

MMA Programming Tips 47

7.1 Mixer and Setup Features

Register Access

The control chip registers are implemented as a set of phantom registers to the second bank of FM registers. Access to the the control chip is triggered by writing 0FFh to the address register of the second FM bank (38Ah). Thereafter, all reads/writes will access the control chip. Access to the second FM bank is returned by writing 0FEh to the same address register.

As with the FM and sampling chips, the control chip uses two port addresses. The first address, 38Ah, is the address register and writing a register number to this address selects a given data register. The second address, 38Bh, is the data address. Values written to this address are directed to the register number specified by the previous write to the address register. There are delays that must be respected when writing to certain registers. These delays are explained in detail in the *Status Register* section.

By default, the control chip is located at 38Ah and 38Bh. However, the chip may be relocated (as explained in the section *Audio Relocalization*). Regardless of where the chip is located, the data register port address is always one greater than the address register port address.

All data registers on the control chip are read/write. Reading a register will return its current value. The only execption to this are registers 0 and 1. All registers are explained below in detail.

The Gold cards contain permanent memory (EEPROM) in which the boot-up values for all registers are stored.

Status Register

Reading the address port (38Ah by default) when the control chip access has been triggered returns the following information:

D7	D6	D5	D4	D3	D2	D1	D0
RB	SB	X	X	SCSI	TEL	SMP	FM

The 4 least significant bits indicate interrupt status. Reading this register does not reset the interrupt status. A zeroed bit indicates which section of the board has generated an interrupt. FM indicates the FM section has generated an interrupt; SMP, the sampling section; TEL, the telephone section; SCSI, the SCSI section. SB set indicates that the card is busy writing to a register. RB set indicates that the card is busy writing its registers to memory.

A delay of approximately 450 µsec is required after writing to any of registers 4 to 8. A delay of approximately 5 µsec is required after writing to any of registers 9 through 16. As well, the chip must not be accessed while the chip is saving its registers to memory. In order to respect these delays, the SB and RB bits should be polled until they become zero. As a general rule, always poll the SB and RB bits before writing anything to the chip.

As well, the chip must not be accessed while it is restoring its registers from memory. This process takes a bit less than 2.5 milliseconds. As there is no status bit for this action, the timing must be done in software.

IMPORTANT: Before returning access to the FM chip (writing FEh to 38Ah), all delays must have expired. Results will be unpredictable otherwise.

Register Map

The diagram on the following page is a summary of the control chip registers. When writing to registers which contain undesignated bits, these bits must be set to zero. Locations where certain bits must be set are indicated by a "1" in the register map.

Register Map, Control Chip

REG	D7	D6	D5	D4	D3	D2	D1	D0
00							ST	RT
01							RING	TC
02	SAMPLING GAIN - LEFT							
03	SAMPLING GAIN - RIGHT							
04	1	1	FINAL OUTPUT VOLUME - LEFT					
05	1	1	FINAL. OUTPUT VOLUME -RIGHT					
06	1	1	1	1	BASS			
07	1	1	1	1	TREBLE			
08	1	1	MU	ST-MONO	SOURCE			
09	FM VOLUME - LEFT							
0A	FM VOLUME - RIGHT							
0B	SAMPLING VOLUME - LEFT							
0C	SAMPLING VOLUME - RIGHT							
0D	AUX VOLUME - LEFT							
0E	AUX VOLUME - RIGHT							
0F	MICROPHONE VOLUME							
10	TELEPHONE VOLUME							
11			SPKR		MFB	XMO	FLT0	FLT1
12								
13	DEN0	DMA SEL 0			AEN	INT SEL A		
14	DEN1	DMA SEL 1						
15		AUDIO RELOCATE						
16	DENS	DMA SEL S			SIEN	INT SEL S		
17		SCSI RELOCATE						
18	SURROUND							

Register Reference

Control/ID

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	X	X	ST	RT

Register #0: Write

Writing to the Control/ID byte with the ST bit set will cause all control chip registers, in their current state, to be written to memory. If RT is set, then all registers will be restored from memory. When the operation is finished, the control chip sets the appropriate bit back to zero. It is not necessary to manually clear the bit.

D7	D6	D5	D4	D3	D2	D1	D0
X	OP2	OP1	OP0	MODEL ID			

Register #0: Read

Reading this register gives information on the model of the card and which options are present. The currently defined MODEL ID's are:

ID	Gold Model
0	2000
1	1000
2	2000MC

The OP0, OP1 and OP2 bits indicate which of the board options are present and are SET when the option is NOT present.

Bit	Option
OP0	Telephone
OP1	Surround
OP2	SCSI

Telephone Control

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	X	X	X	TC

Register #1: Write

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	X	X	RING	TC

Register #1: Read

Setting TC engages the telephone line; clearing the bit hangs up. Reading this register returns the state of the telephone ring signal: RING set indicates that the line is NOT ringing and TC returns the status of the telephone line (i.e. the previously written value of TC).

Sampling Gain

Registers 2 and 3 control the gain on sampling channels 0 (left) and 1 (right). 256 different gain values are possible, giving a range from approximately 0.04 to 10 times the input value. The exact gain is given by the equation:

$$Gain = (RegisterValue * 10) / 256$$

Final Output Volume

These registers control the overall output volume of the card. They replace the potentiometer found on the original Ad Lib card. Adjusting for left and right channels separately allows the balance to be varied.

The volume ranges from +6 dB to -64 dB in steps of 2 dB. An additional step gives -80 dB (off). IMPORTANT: Bits D6 and D7 must be set to 1.

dB	D5-D0
6	3F
4	3E
⋮	⋮
-62	1D
-64	1C
-80	1B
⋮	⋮
-80	0

Registers #4 and #5

Bass

The bass control has a range of +15dB to -12 dB in 3 dB steps. The bass is set using bits D0-D3. IMPORTANT: Bits D4 - D7 must be set to 1.

dB	D3-D0
15	F
⋮	⋮
15	B
12	A
⋮	⋮
0	6
⋮	⋮
-12	2
⋮	⋮
-12	0

Register #6