

	Technical Specification	Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
---	--------------------------------	---

BP30

Midi driver Specification

Edition 2006

Published by Neonseven s.r.l.,
Viale Stazione di Prosecco, 15
34010 Sgonico (Trieste) Italy

© Neonseven.
All Rights Reserved.

For questions on technology, delivery and prices please contact the Neonseven Offices in Italy Sgonico and Gorizia

Attention Please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact Neonseven.

Neonseven technologies may only be used in life-support devices or systems with the express written approval of Neonseven, if a failure of such technologies can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Author	Maurizio Davanzo	Department:	S2	Page: 1/21
Filename	midi_driver_specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –		Confidential	

Table of Contents

1	Document Mission/Scope	3
1.1	Mission	3
1.2	Scope	3
2	List of Acronyms	3
3	Introduction	3
4	Midi Module DSP Interface	4
4.1	Metronome Flag	5
4.2	RMS	5
4.3	Overflow Flag	6
4.4	Headroom	6
5	Audio Postprocessing for Synthesizer	7
5.1	High Frequency Shelving Filter	7
5.2	Audio compressor	8
6	Midi Audio Driver Interface	12
	<i>AUD_ringer_start</i>	12
	<i>AUD_ringer_start_user_tone</i>	13
	<i>AUD_ringer_stop</i>	15
	<i>AUD_ringer_suspend</i>	15
	<i>AUD_ringer_resume</i>	16
	<i>AUD_ringer_stop_suspend</i>	16
7	Midi Audio Driver Implementation Hints	17
8	Test support with Phone Tool	19
8	References	20
8.1	External	20
8.2	Internal	20
9	Document change report	20
10	Approval	20
11	Annex 1	21
12	Annex2	21
13	Annex 3	21

1 Document Mission/Scope

1.1 Mission

This document contains the specification of the internal midi player audio driver.

1.2 Scope

This document is addressed to SW developers who need to interface to internal midi player audio driver.

2 List of Acronyms

Abbreviation / Term	Explanation / Definition
MCU	Micro controller
FS	Sampling Frequency

3 Introduction

In E-GOLDRadio is implemented an internal midi module.

The internal midi player is based on FM synthesis and applies different generators (G0, G1, G2, G3) for the sound synthesis. These generators have different computational complexities and are targeted to different sound synthesis (melodic sounds, drum sounds or other noisy sounds).

The midi module works with 16 KHz or with 32 KHz sampling frequency that will be up-sampled to the 47.649KHz sampling frequency in order to reproduce the sound with the E-GOLDRadio digital to audio converter (DAC).

On E-GOLDRadio up to 40 voices can be played together.

The present version of Midi drivers has been implemented according the following standards:

- General MIDI Lite And Guidelines for Use In Mobile Applications, v1.0, MIDI Manufactures, LA CA 2001
- The Complete MIDI 1.0 Detailed Specification, v96.1, , MIDI Manufactures, LA CA 2001 (file-format 0 and 1)
- Scalable Polyphony MIDI Specification & Profiles, MIDI Manufactures.

Moreover it can play all Imelody files conforming to Imelody v.1.2 specifications.

In what follows, we first discuss in Section 4 the interface between the midi module and the microcontroller. In Section 5 we describe an interface for the midi audio drivers and in Section 6 we provide some hints on how the midi audio drivers have been implemented. In section 7 we discuss the implemented support for test with Phone Tool.

4 Midi Module DSP Interface

This section discusses the interface between the DSP midi module and the micro-controller (MCU).

In order to support the Midi functionality tree new DSP commands have been defined. The new commands are described in **Table 4.1** together with the required parameters.

Every command has been implemented using SYNTH command. Depending on the SWITCH parameter the SYNTH command can have different functions: Switch ON Synthesizer; Switch OFF Synthesizer; Reload a new frame to DSP via shared memory.

To start the Synthesizer, the following sequence has to be used:

- First the first frame has to be send to the DSP with the SYNTH command (SWITCH=2). The frames have a maximum length of 65 words for 20 ms.
- After that, the Synthesizer has to be enabled and the sampling rate has to be set up in the DSP by sending the command SYNTH with the parameter SWITCH set to 1.

During operation of the midi player, the midi module needs 65 word of Midi data every 20ms. So, when the DSP need new data, it genetrates DSP_INT2 Interrupts for the MCU. With every Interrupt, the MCU has to send the next frame to the DSP with the SYNTH command (SWITCH=2). The frame has always to start at the position in the shared memory, where the pointer in the command points to.

When the command SYNTH with the parameter SWITCH set to 2 is received by the DSP, the shared memory location is identified from the PAR2 in the command and then the 65 data words are copied from the shared memory to the internal buffer.

When the end of the file is reached (the last frame is sent to the DSP), the SYNTH command with switch 0 has to be sent to stop the Synthesizer.

Table 4.1: The midi DSP commands

Command	Parameters	Operation and Command Description
DspStartMidiPlayer (SYNTH command)	switch parameter =1	Switch on and init Synthesizer.
	PAR1	Must contain the right value to specify the sample rate (only 16kHz and 32kHz are possible). Before the Synthesizer is switched on it has to be guaranteed the first frame has been copied to the internal memory using SWITCH=2.
	PAR2	Pointer to the shared memory address, where the first input frame is stored.

DspStopMidiPlayer (SYNTH command)	switch parameter =0	Switch off Synthesizer.
	PAR1	Not used.
	PAR2	Not used.

DspReloadMidiPlayer (SYNTH command)	switch parameter =2	Feed a new frame (max. 65 words, 20ms frame based) to DSP via shared memory.
	PAR1	Contains the Headroom of the Synthesizer, in detail this means the amount of right shifts the synthesizer will apply before summing up the output data of all voices to prevent clipping (0 <= Headroom <= 16).
	PAR2	Not used.

Each Synthesizer run provides 5 ms output and passes the samples to the Circular Buffer. The data frames provided by the MCU correspond to 20 ms output. Therefore the synthesizer is called four times to process one MCU data frame.

Every 20 ms the midi module reads and interprets the input (MIDI) data for one frame from the input buffer, writes generated samples to the Circular Buffer, and returns three status words:

- Synthesizer Status (bit 0: Overflow Flag, bit 1: Metronome Flag, other bits currently unused)
- Low RMS
- High RMS

They have to be read by the MCU before writing new MIDI data.

4.1 Metronome Flag

The Metronome Flag indicates that a music beat (by default ¼ Note beat, can be changed via MIDI time signature message) has occurred in the current 20 ms frame. If this flag is copied to e.g. a LED, it will flash synchronous to the beat of the music played.

4.2 RMS

High and Low RMS are logarithmic RMS values calculated over 20ms of audio data with a resolution of 1/16dB. The maximum value is 1500 (corresponding to 0dB). The high and low channels are derived by applying a

Author	Maurizio Davanzo	Department:	S2	Page:	5/21
Filename	midi_driver_specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

simple first order FIR low-/highpass filter before the RMS calculation. They can be used for generation of light/vibro effects in addition to the metronome signal.

4.3 Overflow Flag

The Overflow Flag indicates whether clipping occurred during sample generation:

- Overflow Flag = 1 (overflow occurred)
- Overflow Flag = 0 (overflow not occurred)

The user could use this information to implement a simple volume control or some kind of compression algorithm. The basic rules for such an algorithm could be:

- Every time Overflow Flag = 1 (overflow occurred), the headroom should be decreased by one. This would correspond to a decrease of the output amplitude by 6 dB.
- Every Time Overflow Flag = 0, the headroom can be increased by one for the next call.

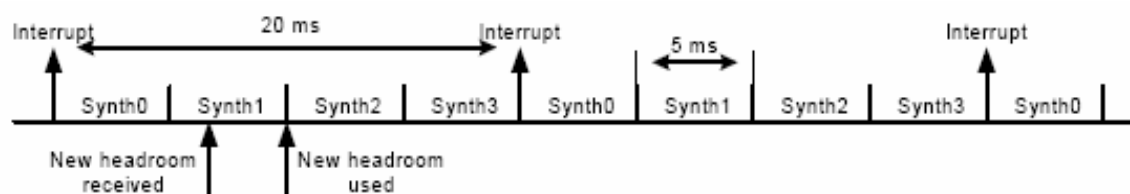
It is to say that this simple algorithm might need some kind of “low-pass-filtering” to avoid annoying fluctuations of the headroom value from frame to frame (in one frame headroom might be decreased, in the next it might be increased, and so on). Also the rough resolution (6 dB) of the headroom parameter might introduce some audible clicks to the output signal if headroom was changed from one frame to the next. Inside present version of Midi drivers the headroom algorithm has not been implemented and the headroom value is fixed to 3.

4.4 Headroom

Headroom is the number of applied right shifts for each internal generator output. When overflow (clipping) occurred during summing up of the output data of all voices. The headroom should be increased.

Every 20m sec the Synthesizer writes the Synthesizer parameters to the shared memory location SM_SYNTH_PAR_OUT. Using these parameters the MCU decides, whether the headroom should be increased or not via the parameter Headroom in the SYNTH Command (SWITCH=2). If the MCU sends a new headroom value to the DSP, this value is used for the next Synthesizer run even if the previous is still being processed (See **Figure 4.1**).

Figure 4.1 Timing diagramm for Synthesizer



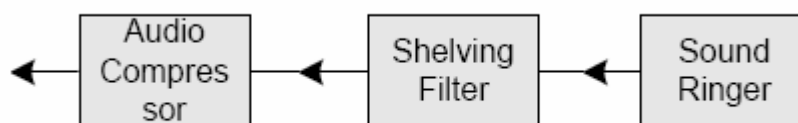
Inside present version of Midi drivers the headroom algorithm has not been implemented and the headroom value is fixed to 3.

5 Audio Postprocessing for Synthesizer

The Output samples of the synthesizer are postprocessed by two modules:

- Hi Frequency Shelving Filter
- Audio Compressor

Figure 5.1 **signal path**



5.1 High Frequency Shelving Filter

This module is implemented as a first order IIR Filter, which is mainly used for high frequency boost in audio signals.

Its transfer function is given by:

$$H_{SZH}(z) = \frac{b_0 \cdot 2^{b_exp} + b_1 \cdot 2^{b_exp} \cdot z^{-1}}{1 + a_1 \cdot z^{-1}}$$

where b_exp , b_0 , b_1 and


a_1 are the filter

coefficients set by the AUDIOPOSTPROC command.

This command is described in **Table 5.1** with the required parameters.

Table 5.1: AUDIOPOSTPROC command

AUDIOPOSTPROC	Switch on/off the high frequency shelving filter and change its coefficients.	
	AudioPostProcSwitch	Bit_0: Switch off/on the shelving filter 0: off / 1: on (Default 1, shelving filter active)
	b_exp b1 b0	High Frequency Shelving Filter Coefficients (See Table 3 for default values at 16 and 32 kHz)

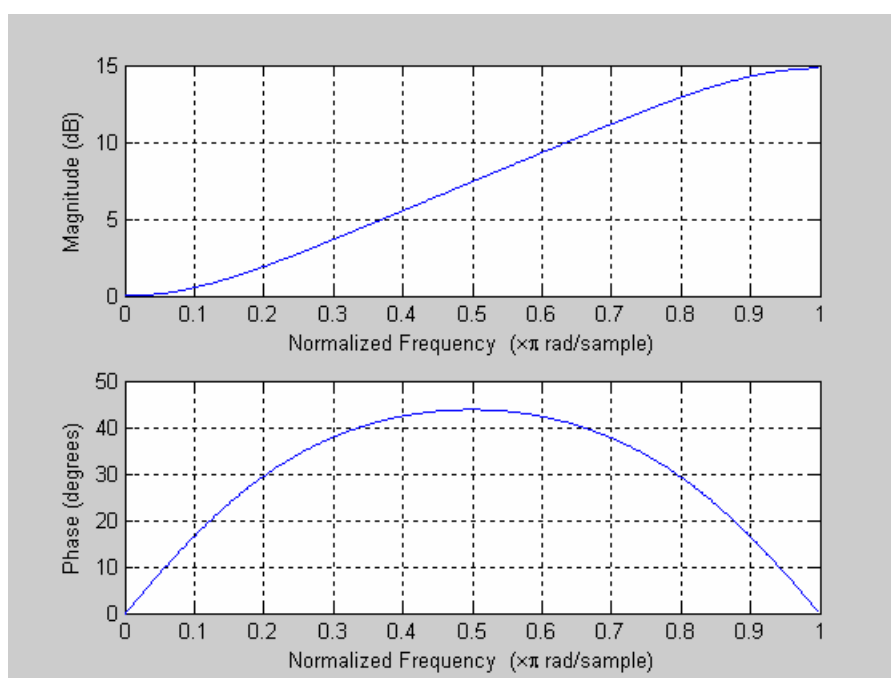
	Technical Specification		Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
	a1		

The default settings for four coefficients of the High Frequency Shelving Filter (at 16 kHz) are listed in **Table 5.2**; in **figure 5.1** is showed the corresponding Frequency Responses.

Table 5.2: Default Settings for filter coefficients

Parameter Name	Default Value at 16 kHz
b_exp 2	2
b1	0xE19A
b0	0x4B33
a1	0x3333

Figure 5.2 Frequency Responses



5.2 Audio compressor

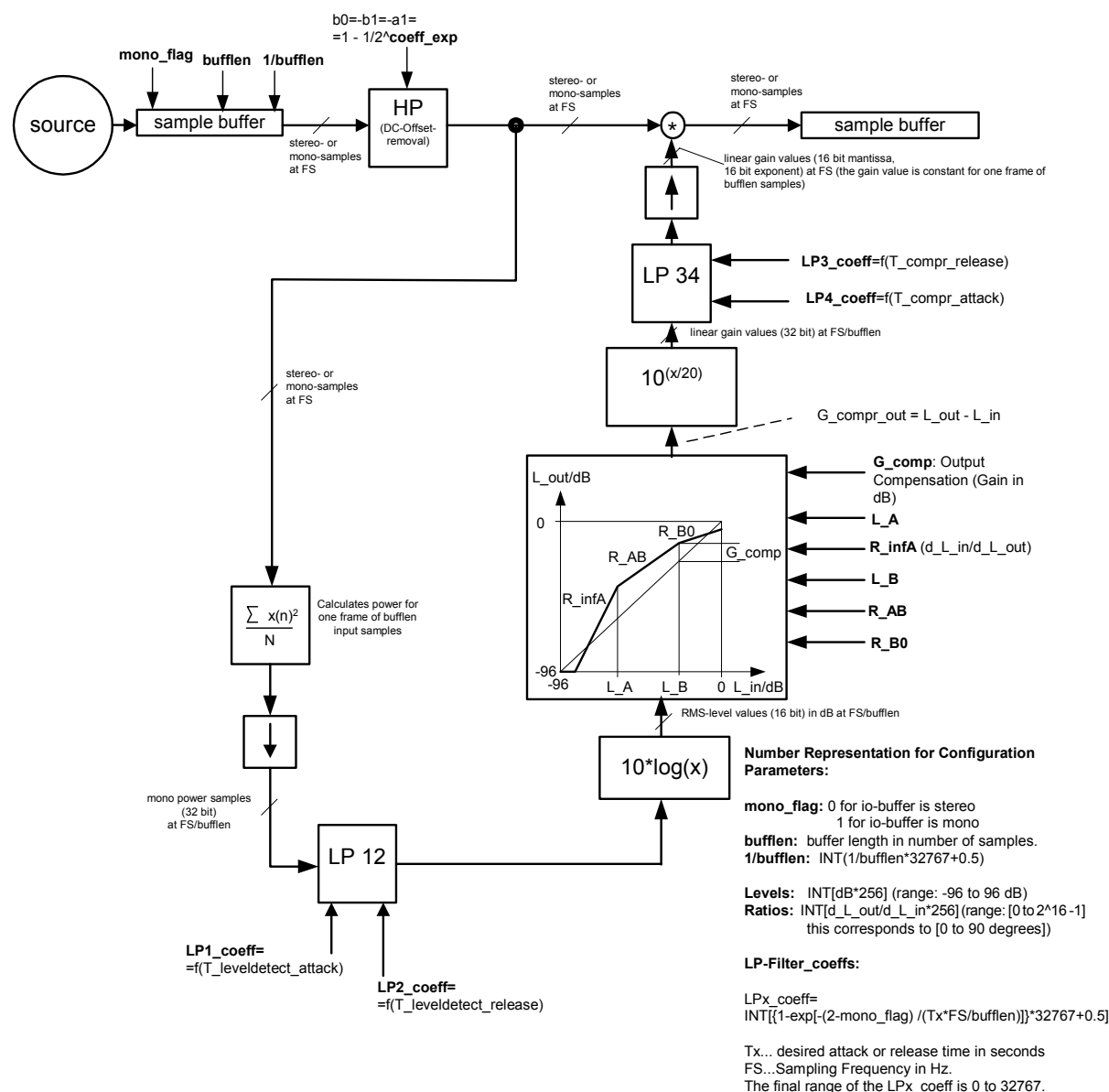
The audio compressor is a device for manipulating the dynamic range of audio signals.

Author	Maurizio Davanzo	Department:	S2	Page:	8/21
Filename	midi_driver_specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	

Error! Reference source not found. **5.3** shows the block diagram of the Audio compressor. The samples from the audio source are fed block wise into the compressor. After a high pass filter, which removes a possible DC-offset in the source signal, for each block of input samples, a rms-level measurement is performed (mean of input powers, root is taken into account during transformation to log-domain: $10 \cdot \log(x)$). According to the adjusted compressor curve settings (14 parameters), the corresponding output gain is estimated, which is then applied to the input-audio samples (the output gain is constant for one input-output-block!).

For the gain measurement and the output level, attack- and release-times can be adjusted by means of the low-pass filters LP_12 and LP_34. As the level estimation is performed block wise, the corresponding calculations run at a lower sampling frequency $FS/\{\text{bufflen}/(\text{mono_flag}+1)\}$.

Figure 5.3 Blok diagram of the Audio Compressor



Author	Maurizio Davanzo	Department:	S2	Page:	9/21
Filename	midi_driver_specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

	Technical Specification	Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
---	--------------------------------	---

The audio compressor is controlled by 14 configuration parameters. **Table 5.2** gives an explanation for each parameter and **Table 5.3** shows the default values.

Table 5.2: Description of 14 parameters for the Audio Compressor

Author	Maurizio Davanzo	Department:	S2	Page: 10/21
Filename	midi_driver_specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

Parameter Name (all in 16 bit format)	Range and Number Representation	Description
mono_flag	0: if io-buffer is stereo 1: if io-buffer is mono	Flag, which indicates if input buffer is stereo or mono. (the maximal valid length is limited by the define)
m_bufflen	Range: 1 to 32767	Length of io-buffer in number of samples.
m_inv_bufflen	Range: 1 to 32767	Inverse of buffer length in Q1.15 representation.
m_hp_coeff_exp	Range: 0 to 15	Determines coefficients for HP-Filter, which is used for DC-removal: $b_0 = -b_1 = -a_1 = 1 - 1/2^{\text{coeff_exp}}$ $H(z) = \frac{b_0 + b_1 z^{-1}}{1 - a_1 z^{-1}}$
m_lp1_coeff	0 to 32767	Filter coefficient determining the attack time of the rms-level measurement.
m_lp2_coeff	0 to 32767	Filter coefficient determining the release time of the rms-level measurement.
m_lp4_coeff	0 to 32767	Filter coefficient determining the attack time of the compressor.
m_lp3_coeff	0 to 32767	Filter coefficient determining the release time of the compressor.
m_L_A	-96*256 to 0	Lower compressor input threshold. Must fulfill $m_L_A < m_L_B$
m_L_B	-96*256 to 0	Upper compressor input threshold. Must fulfill $m_L_A < m_L_B$
m_G_comp	-96*256 to 96*256	Output compensation gain: Sets output level of L_B: $L_B_out = L_B + G_comp$
um_R_infA	0 to $(2^{16} - 1)$	Gradient of compression curve for input levels in the range of -96 dB to L_A. (Only positive gradients between 0 and 90 degrees are possible).
um_R_AB	0 to $(2^{16} - 1)$	Gradient of compression curve for input levels in the range of L_A to L_B. (Only positive gradients between 0 and 90 degrees are possible).
um_R_B0	0 to $(2^{16} - 1)$	Gradient of compression curve for input levels in the range of L_B to 0dB. (Only positive gradients between 0 and 90 degrees are possible).

Table 5.3: Default Settings for Audio Compressor

Parameter Name	Default Value at 16 kHz	Default Value at 32 kHz
INITDATA_mono_flag	0	0
INITDATA_m_bufflen	40	80
INITDATA_m_inv_bufflen	819	410
INITDATA_m_hp_coeff_exp	6	7
INITDATA_m_lp1_coeff	20713	20713
INITDATA_m_lp2_coeff	1985	1985
INITDATA_m_lp4_coeff	12893	12893
INITDATA_m_lp3_coeff	809	809
INITDATA_m_L_A	-7680	-7680
INITDATA_m_L_B	-2560	-2560
INITDATA_m_G_comp	512	512
INITDATA_um_R_infA	256	256
INITDATA_um_R_AB	128	128
INITDATA_um_R_B0	0	0

6 Midi Audio Driver Interface

In this Section a synthetic description is given for the interface functions of midi audio driver.

The midi module introduces a novel resource that has been added to the resource table, the resource is called: `aud_resource_midi_player`. We deal with this resource in a manner absolutely similar to the other resources by using the function `AUD_allocate_resource()` in order to reserve it and `AUD_release_resource()` in order to release it.

The following interface functions have been implemented:

AUD_ringer_start

Prototype:

SINT8 `AUD_ringer_start`(UINT8 `handle`, `aud_ringer_id_enum` `tone_id`, UINT16 `nof_repeats`, `aud_ringer_device_enum` `device`);

Parameters:

UINT8 `handle`

`aud_ringer_tone_id_enum` `tone_id` (depends on project, found in `aud_drv.h` file)

	Technical Specification	Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
---	--------------------------------	---

UINT16 nof_repeats

aud_ringer_device_enum device

Returns: see standard return parameters in [1].

Description: Plays the requested midi or imelody melody. The nof_repeats indicates how many times the tone shall be repeated. Setting this parameter to 0, means repeat forever. If a finite number (i.e. a number different from 0) is selected, the audio driver sends a message back to the calling module when the midi melody is finished. This does not release the midi player resource. The normal release procedure for resources must still be followed. The device parameter told whether sound and/or vibrator should be used.

The volume level is the last one selected by AUD_set_volume function.

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_ringer_tone_finish	If the nof_repeats parameter is different from 0, the AUD_DRIVER_RSP signal with this return code is sent when the midi melody has played to the end. If nof_repeats is 0 the midi player must be stopped by the function AUD_midi_stop(), this does not result in sending the SDL signal.
aud_rc_midi_conv_error	If an error is found in data while opening or parsing the midi file, the midi player is stopped and the AUD_DRIVER_RSP signal with this return code is sent back to the calling process. The resource is still allocated.

AUD_ringer_start_user_tone

Prototype:

SINT8 AUD_ringer_start_user_tone(UINT8 handle, UINT16 huge *ringer_data, UINT16 nof_repeats, UINT32 size, aud_ringer_tone_format_enum format, aud_ringer_device_enum device, UINT8 channel, UINT8 channel_volume);

Parameters:

UINT8 handle

UINT16 huge *ringer_data

UINT16 nof_repeats

UINT32 size

aud_format_enum format

aud_ringer_device_enum device

Author	Maurizio Davanzo	Department:	S2	Page: 13/21
Filename	midi_driver_specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

	Technical Specification	Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
---	--------------------------------	---

UINT8 channel

UINT8 channel_volume

Returns: see standard return parameters in [1].

Description: Plays a user defined melody. Different parsers have been written for the different melody formats in order to directly convert the audio content in the format accepted by the DSP at play time. If the audio driver finds a syntax error in the melody data, a SDL signal with an error code is sent back to the calling procedure. The `nof_repeats` indicates how many times the tone shall be repeated. Setting this parameter to 0, means repeat forever. If a finite number (i.e. a number different from 0) is selected, the audio driver sends a message back to the calling module when the midi melody is finished. This does not release the midi player resource. The normal release procedure for resources must still be followed. The Vibrator is controlled independently of the midi player. The volume level is the last one selected by `AUD_set_volume` function. The channel and channel_volume parameter aren't supported and they will be only used for SMAF-phrase format, i.e. for all other formats these parameters are don't care.

Enum: aud_format_enum	
Enum name	Description
aud_format_midi	MIDI comprises many different formats. Only the following formats are supported: SMF format 0 and 1 and SP-MIDI
aud_format_imelody	Any Imelody version 1.2, class 1.0 file is supported
aud_format_end	

Enum: aud_ringer_device_enum	
Enum name	Description
aud_ringer_device_sound	Only poly ringer
aud_ringer_device_vibrator	Only poly vibro
aud_ringer_device_sound_and_vibrator	Both poly ringer and vibro
aud_ringer_device_end	

Return codes (sent in AUD_DRIVER_RSP signal)

Author	Maurizio Davanzo	Department:	S2	Page:	14/21
Filename	midi_driver_specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	

Return code	Comment
Aud_rc_midi_player_finish	If the <code>nof_repeats</code> parameter is different from 0, the AUD_DRIVER_RSP signal with this return code is sent when the midi melody has played to the end. If <code>nof_repeats</code> is 0 the midi player must be stopped by the function <code>AUD_midi_stop()</code> , this does not result in sending the SDL signal.
aud_rc_midi_conv_error	If an error is found in data while opening or parsing the midi file, the midi player is stopped and the AUD_DRIVER_RSP signal with this return code is sent back to the calling process. The resource is still allocated.

AUD_ringer_stop

Prototype:

SINT8 AUD_ringer_stop(UINT8 handle, UINT8 channel);

Parameters:

UINT8 handle

UINT8 channel

Returns: see standard return parameters in [1].

Description:

Immediately stops and power down the ringer device. The channel parameter will be supported only with SMAF-Phrase format.

AUD_ringer_suspend

Prototype:

SINT8 AUD_ringer_suspend(UINT8 handle, UINT8 SlotID, UINT8 channel);

Parameters:

UINT8 handle

Author	Maurizio Davanzo	Department:	S2	Page:	15/21
Filename	midi_driver_specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

	Technical Specification	Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
---	--------------------------------	---

UINT8 SlotID [0..4]

UINT8 channel [0..4]

Returns: see standard return parameters in [1].

Description:

This function suspends all ongoing activities on the internal midi player. The internal midi player is restarted using the `AUD_ringer_resume` function.

After the midi play suspension, a new melody can be activated.

`SlotID` (from 0 to 4) is used for nesting up to 5 midi melodies.

The audio driver expects the MMI to indicate which `SlotID` is supposed to be used.

If the MMI sends an `AUD_midi_suspend` command requesting a `SlotID` that already stores a suspended melody that has not been resumed, the audio driver will overwrite this “old” suspended melody without generating any error message.

The channel parameter is not supported.

AUD_ringer_resume

Prototype:

SINT8 `AUD_ringer_resume`(UINT8 handle, UINT8 SlotID, UINT8 channel);

Parameters:

UINT8 handle

UINT8 SlotID

UINT8 channel

Returns: see standard return parameters in [1].

Description:

It resumes the midi melody that was suspended with the `AUD_ringer_suspend` function.

`SlotID` (from 0 to 4) is used for nesting up to 5 midi melodies.

The channel parameter is not supported.

Return codes (sent in `AUD_DRIVER_RSP` signal)

Return code	Comment
<code>Aud_rc_midi_resume_error</code>	If the <code>SlotID</code> does not point to any suspended midi melody, this return code is sent.

AUD_ringer_stop_suspend

Author	Maurizio Davanzo	Department:	S2	Page: 16/21
Filename	midi_driver_specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

Prototype:

SINT8 AUD_ringer_stop_suspend(UINT8 handle, UINT8 SlotID)

Parameters:

UINT8 handle

UINT8 SlotID

Returns: see standard return parameters in [1].

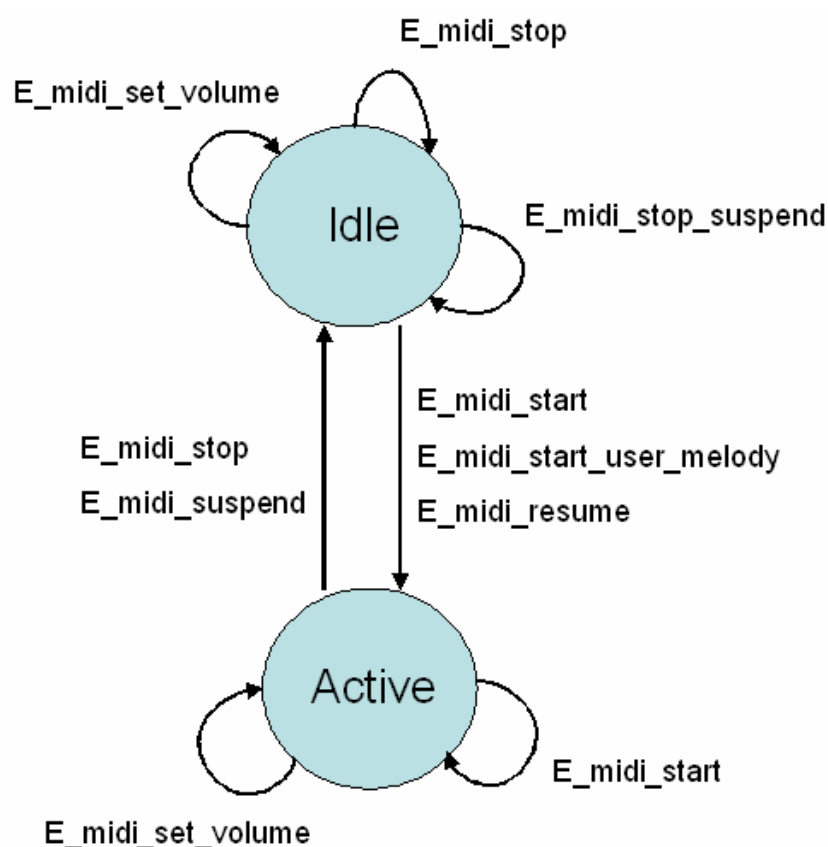
Description:

Deletes information for a suspended midi melody in the suspend nest queue.

7 Midi Audio Driver Implementation Hints

The midi player state machine has been implemented according to **Figure 6.1**.

Figure 6.1: Midi Player state machine



	Technical Specification	Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
---	--------------------------------	---

If the `AudMidiPlayerSM` is in IDLE state and it receives a `E_Midi_Start` event (or a `E_midi_start_user_melody` event), the midi player state machine opens the selected midi/lmelody file, checks it and initializes and activates the midi engine controller at the uC side. This midi engine controller writes to the shared memory a first data frame and activates the midi player at the DSP side with the SYNTH command (SWITCH=1). After that, new midi data are passed when the DSP generates `DSP_INT2` Interrupts for the MCU. With every Interrupt, the MCU has to send the next frame to the DSP with the SYNTH command (SWITCH=2).

The `HandleMidiPlayerTimeout()` (called by L1 every TDMA frame) implements the midi controller. It takes care of the DSP midi player activation.

Every 20ms the interrupt service routine (`MIDI_Irq_Proc()`) writes 65 words of midi data to the shared memory. This function must not be computational intensive, only dummy parameter passing has to be performed by this routine. On the contrary, the midi parsing and conversion to the data format accepted by the midi DSP module is performed by a lower priority task (i.e. the `OS_PROCESS(AUD_highisr)`).

For this purpose two ping-pong buffers has been introduced. Each of these buffers memorizes 5 arrays of 65 words, i.e. they buffer 100 ms of midi data. One buffer is used by the `MIDI_Irq_Proc()` procedure for reading the midi data that is passed to the DSP. The other buffer instead is used by the `AUD_highisr` task for writing the midi data obtained during conversion of the midi/lmelody file.

The `MIDI_Irq_Proc()` every 100ms checks that the `AUD_highisr` task has completed the conversion, exchanges the role of the two buffer pointers and activates again the parser by setting the `audSemaPID`.

If for some reason the `AUD_highisr` task does not complete its job in the 100-200 ms period, the midi player simply prolong the current sound by multiples of 20ms in order to provide more time for conversion in the `AUD_highisr` task.

In summary the midi player audio drivers are split into three priority levels:

- Audio interface level, where the audio dispatcher and the `AudBaseSM` are processed. This level has the lowest priority and is used only for initialization, activation or deactivation purposes.
- Parser level, located in `AUD_highisr` task. Parsing and conversion is activated every 100-200 ms by the midi engine controller. This level has a medium priority.
- TDMA frame level and interrupt level, where the `HandleMidiPlayerTimeout()` and `MIDI_Irq_Proc()` procedure are executed. During midi player activity, every 20 ms 65 words of midi data are written to shared memory and every 100 ms the `audSemaPID` are activated for initiating a novel midi conversion cycle.

A suspend and resume procedure similar to that already available for the external polyringer has also been developed. The suspend procedure simply consists in stopping the midi player and memorizing any information necessary for resuming the activity from the same position that was reached (particularly, it is memorized the pointer to the melody, the size of the melody, the repeat information, and the index of the current event being processed).

When resuming a midi file, the midi parser chaise the file from the beginning up to the last event processed before suspend: all note-on and note-off events are skipped, all control events are sent one after the other to the DSP in order to obtain the same status of the midi player before suspend. Particularly, in order to reduce as much as possible the delay before resuming the melody, all control events have been divided in two groups: one group is constituted by rare control events, which the parser sends immediately to the DSP without memorizing the corresponding status information; the second group is constituted by common control commands (like pitch bend, volume change, etc.), which are repeatedly issued by the midi file. In this case, only the last control event of each control function affects the sound generation after resume. Therefore, the parser memorizes the corresponding status variation without sending this command to the DSP. Once reached the last event processed before suspend, the parser eventually tunes the midi player internal status by issuing the last control change command for each control function. From this moment on the midi player continue to work in a normal manner.

Author	Maurizio Davanzo	Department:	S2	Page: 18/21
Filename	midi_driver_specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential

	Technical Specification	Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
---	--------------------------------	---

When resuming an Imelody file, the Imelody parser chaises the file from the beginning up to the last event processed before suspend: all note events are skipped while all volume events are sent one after the other to the DSP in order to obtain the same output volume before the suspend action.

A volume normalization procedure has also been introduced both for midi and Imelody files. When opening the melody file, the parser first scan the entire file in order to determine the maximum volume selected in the file (in midi the maximum channel volume and the maximum global volume). When playing, all volume commands will be normalized in order to get the maximum volume at output of the midi player. In this way, the volume uncertainty due to the melody score is removed. The volume can still be set to the desired level by acting on the loudspeaker gain stages.

It must be pointed out that Imelody, due to its monophonic content, has a volume 12-18 dB lower than midi. This volume difference shall be recovered by acting on the “headroom” bits of the midi player control or by acting of the loudspeaker gain stages.

In order to flash LED synchronous to the beat of the music played or to the RMS values, every 20ms the interrupt service routine (MIDI_Irq_Proc ()) calls function LED_configure_midi () that checks the status words: Synthsizer Status and Low/High RMS to turn on/off LEDs.

8 Test support with Phone Tool

Support for testing the midi player with Phone Tool has also been implemented. Particularly, the user commands reported in **Table 7.1** have been implemented.

Table 7.1 The midi DSP commands

Command	Parameters	Operation and Command Description.
50	0	Allocate the midi player resource.
51	0	Release the midi player resource.
52	2	Call AUD_ringer_start with tone_id = %1 and repeat = %2
	%1	tone_id
	%2	repeat
53	0	Call AUD_ringer_start_user_tone. It plays some predefined test midi/imelody files.
	%1	Index of selected midi/imelody file
	%2	Repeat information
54	0	Call AUD_ringer_start_user_tone. It plays some predefined test midi/imelody files.
	%1	Index of selected midi/imelody file
	%2	Repeat information

Author	Maurizio Davanzo	Department:	S2	Page:	19/21
Filename	midi_driver_specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	

55	0	Stop midi player
----	---	------------------

56	0	Call AUD_ringer_suspend.
	%1	Selected suspend slot

57	0	Call AUD_ringer_resume
	%1	Selected suspend slot

58	0	Call AUD_ringer_stop_suspend
	%1	Selected suspend slot

8 References

8.1 External

- [2] "E-GOLDlite Firmware Manual G14 Mask", Rev. 1.02, 2004-09-10
- [3] "High Frequency Shelving Filter", Rev. 01, 2004-05-11
- [4] "Audio Compressor: Description of the Configuration Parameters for the Audio Compressor", Rev. 04, 2004-05-11

8.2 Internal

Title	Doc ID
[1] Audio driver specification	AH01.SW.TS.000021

9 Document change report

Rev	Change Reference		Record of changes made to previous released version	
	Date	CR	Section	Comment
1.0	21/06/2004	none	Document created	
1.1	29/04/2005	none	Document updated to BP2.1	
1.2	31/01/2006	none	Document updated to BP30 Platform	

10 Approval

Revision	Approver(s)	Date	Source/signature
1.0	Stefano Godeas	21/06/2004	Document stored on server
1.1	Stefano Godeas	29/04/2005	Document stored on server
1.2	Stefano Godeas	31/01/2006	Document stored on server

	Technical Specification	Doc. ID: AH01.SW.TS.000022 Rev.:1.2 Date:31/01/2006
---	--------------------------------	---

11 Annex 1

None

12 Annex2

None

13 Annex 3

None

Author	Maurizio Davanzo	Department:	S2	Page: 21/21
Filename	midi_driver_specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential