NEONSEVEN

Midi Player

Maurizio Davanzo maurizio.davanzo@neonseven.com

NEONSEVEN

Introduction

The present version of Midi drivers can support the following file format:

- **Standard MIDI file - type 0 and 1**
- **MIDI Lite - type 0**
- **SP MIDI**
- **Imelody file**

The Midi drivers has been implemented according the following standards:

- **The Complete MIDI 1.0 Detailed Specification**, v96.1, , MIDI Manufactures, LA CA 2001 (file-format 0 and 1)
- General MIDI Lite And Guidelines for Use In Mobile Applications, v1.0, MIDI Manufactures, LA CA 2001
- **Scalable Polyphony MIDI Specification & Profiles**, MIDI Manufactures.
- Imelody v.1.2 specifications

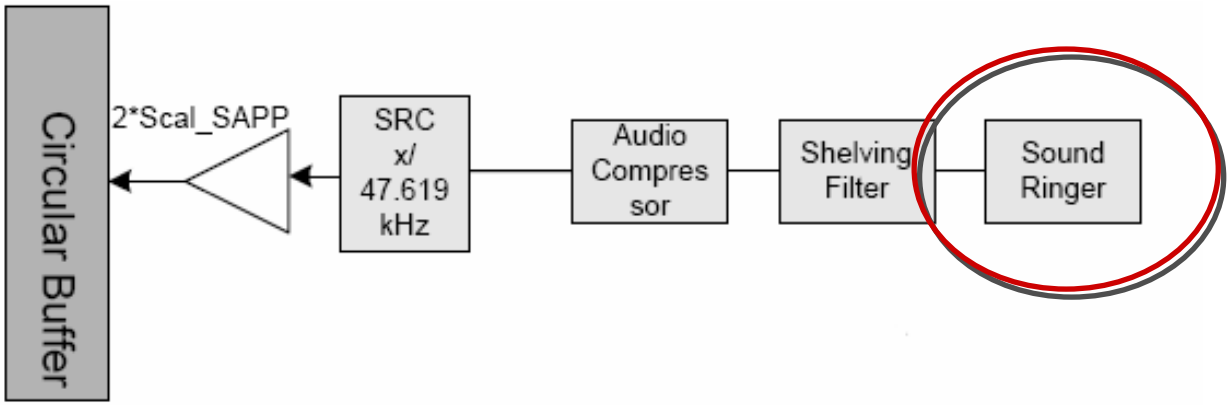
NEONSEVEN

MIDI MODULE

In **E-GOLDlite firmware mask G14** is implemented an internal **midi module**:

- It is based on FM synthesis and applies 4 different generator types
- It works with 16 KHz or with 32 KHz sampling frequency
- It can play up to 40 voices together

Voiceband Processing System:



MIDI MODULE - interface (SYNTH cmd)

The interface between the DSP midi module and the micro-controller has been implemented by **SYNTH command**. Depending of the **SWITCH** parameter the command can have different functions:

- DspStartMidiPlayer: SYNTH cmd with SWITCH=1

  PAR1:sample rate

- DspStopMidiPlayer: SYNTH cmd with SWITCH=0

- DspReloadMidiPlayer: SYNTH cmd with SWITCH=2

  PAR1:Headroom **PAR2**:Pointer to the shared memory address

NEONSEVEN

MIDI MODULE - interface (SYNTH cmd)

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

1. First the first frame of midi data has to be send to the DSP with the SWITCH=2. (The frames have a maximum length of 65 words for 20 ms)
2. After that, the Synthesizer has to be enabled and the sampling rate has to be set up in the DSP by sending the command SWITCH=1.
3. The DSP will then generate DSP interrupts for the MCU to get new frames.
4. With every Interrupt, the MCU has to send the next frame to the DSP with the SYNTH cmd (SWITCH=2).
5. When the end of the file is reached the SYNTH cmd with the SWITCH=0 has to be sent to stop the Synthesizer.

NEONSEVEN

MIDI MODULE - interface (SYNTH cmd)

With every interrupt the DSP return to the MCU three words:

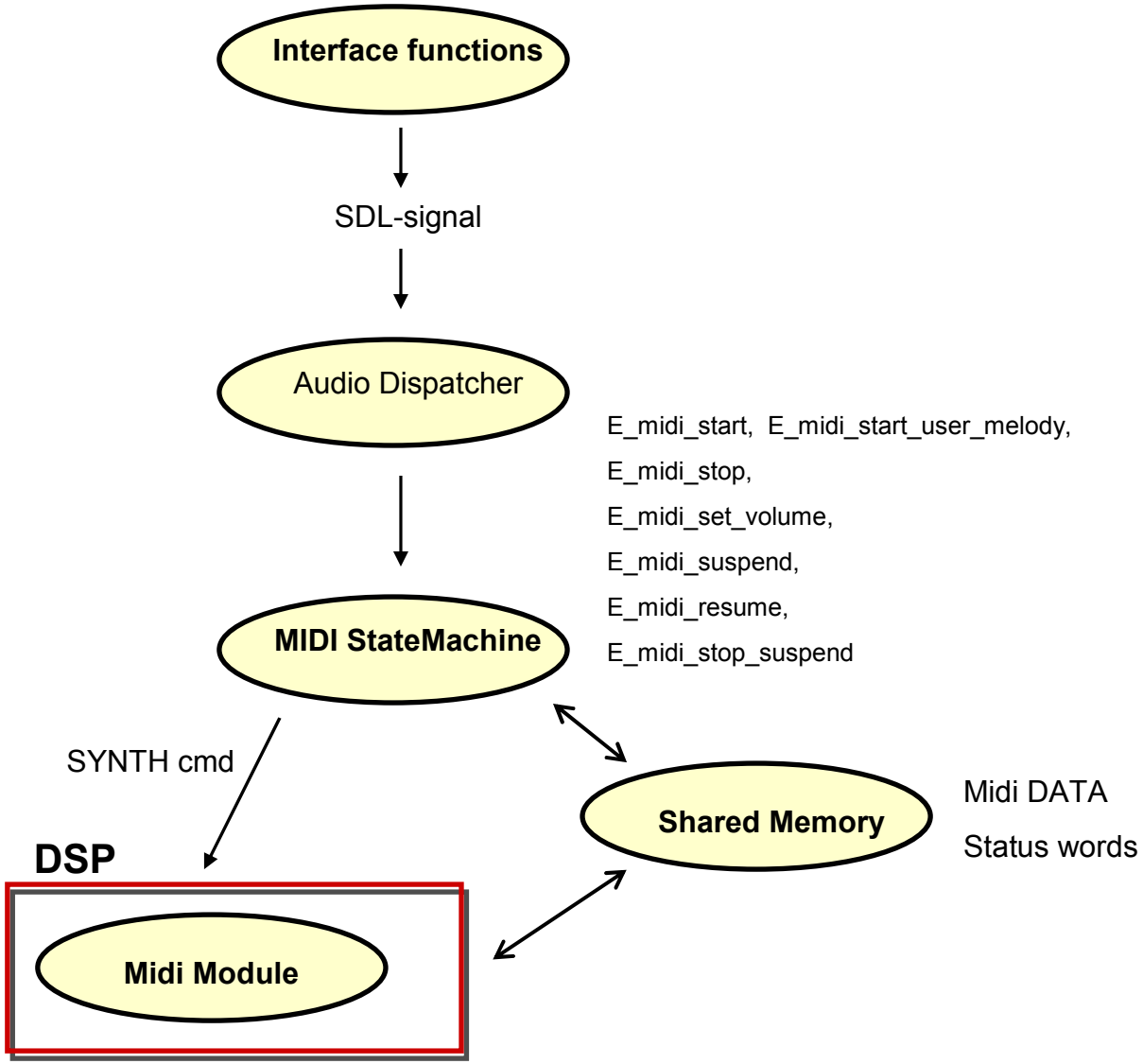
- **Status word**

- Bit 0 (Limit flag): Overflow (clipping) occured during summing up of the output data of all voices in the last frame. (The Headroom should be increased)
- Bit 1 (Metronom flag): Metronom-Click occured in current frame

- **RMS low**: RMS vaue of low frequency part of Synthesizer output in last frame
- **RMS high**: RMS vaue of high frequency part of Synthesizer output in last frame

NEONSEVEN

MIDI Driver - Overview



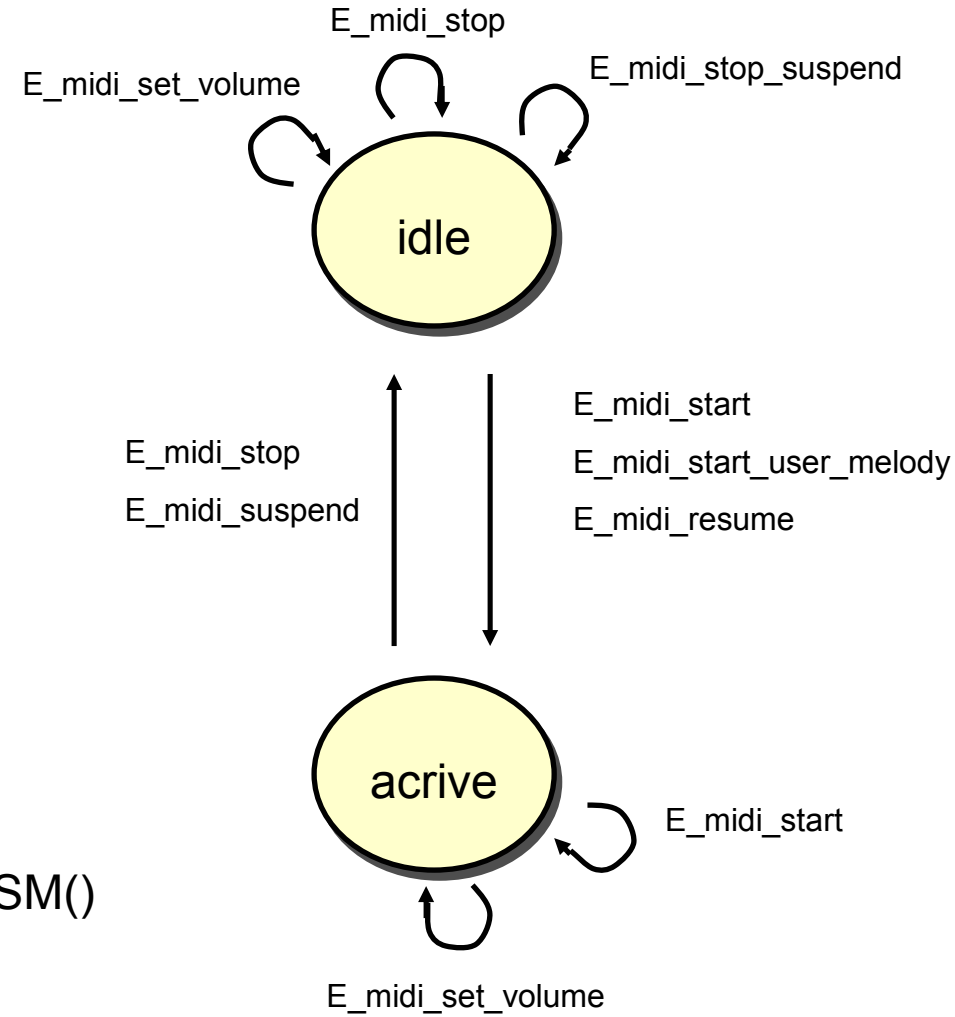
NEONSEVEN

Midi Driver - Interface

- **AUD_midi_start**(UINT8 handle, aud_ringer_id_enum tone_id, UINT16 nof_repeats, aud_ringer_device_enum device);
- **AUD_midi_start_user_melody**(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);
- **AUD_midi_stop**(UINT8 handle, UINT8 channel);
- **AUD_midi_suspend**(UINT8 handle, UINT8 SlotID);
- **AUD_midi_resume**(UINT8 handle, UINT8 SlotID)
- **AUD_midi_stop_suspend**(UINT8 handle, UINT8 SlotID)

NEONSEVEN

Midi Driver – State Machine



The State Machine has been implemented in function:

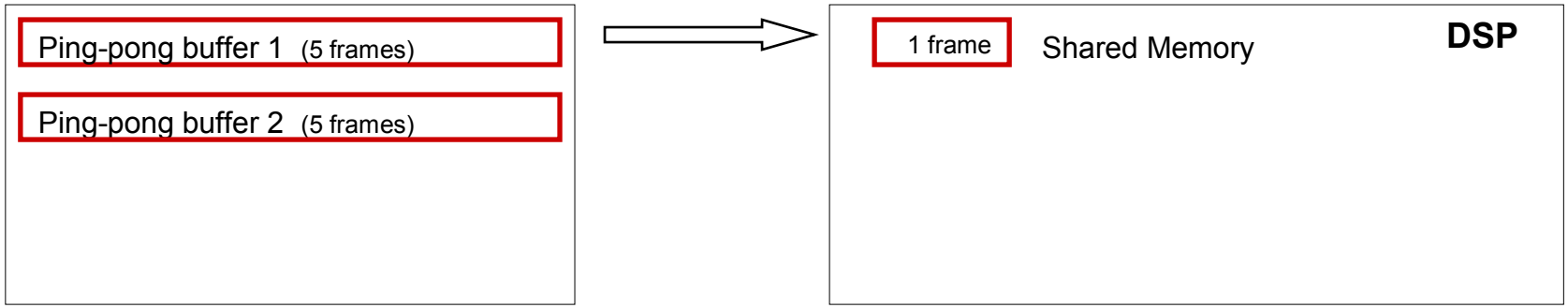
AudInternalMidiPlayerSM()

(it is initialized in Idle)

Midi Driver – **start**

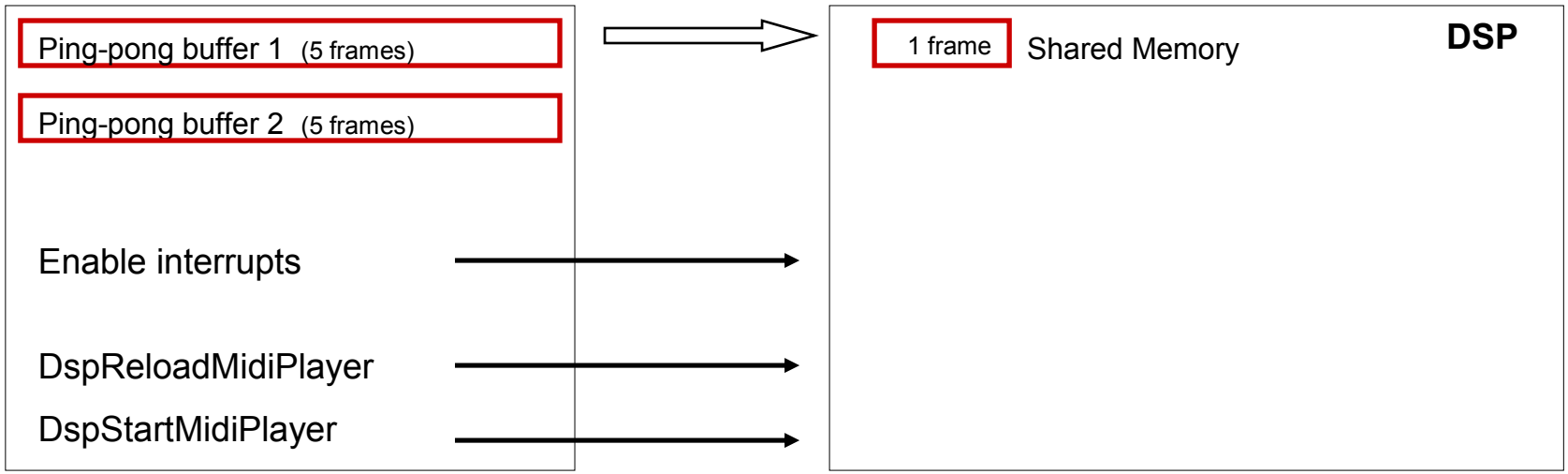
when the state machine recive a **E_Ringer_Start** event, it performs the following activity:

- ✓ **Set “aud_resouce_midi_player”**
- ✓ **Open the audio path**
- ✓ **Check the midi file**: reads the Header Chunk and checks the Track Chunk size.
- ✓ **Normalize the midi file volume** (see function “NormalizeMidiVolume”)
- ✓ **Initialize the two ping-pong buffers**
- ✓ **Translate the first 5 frames of midi data and put it inside the ping-pong buffer 1**
- ✓ **Copy the first frame into Shared Memory**



Midi Driver – **start**

- ✓ **Send command** “DspReloadMidiPlayer”
- ✓ **Set the StateMachine to <u>Active</u>**
- ✓ **Enable interrupts from DSP to MCU**
- ✓ **Send command** “DspStarMidiPlayer”



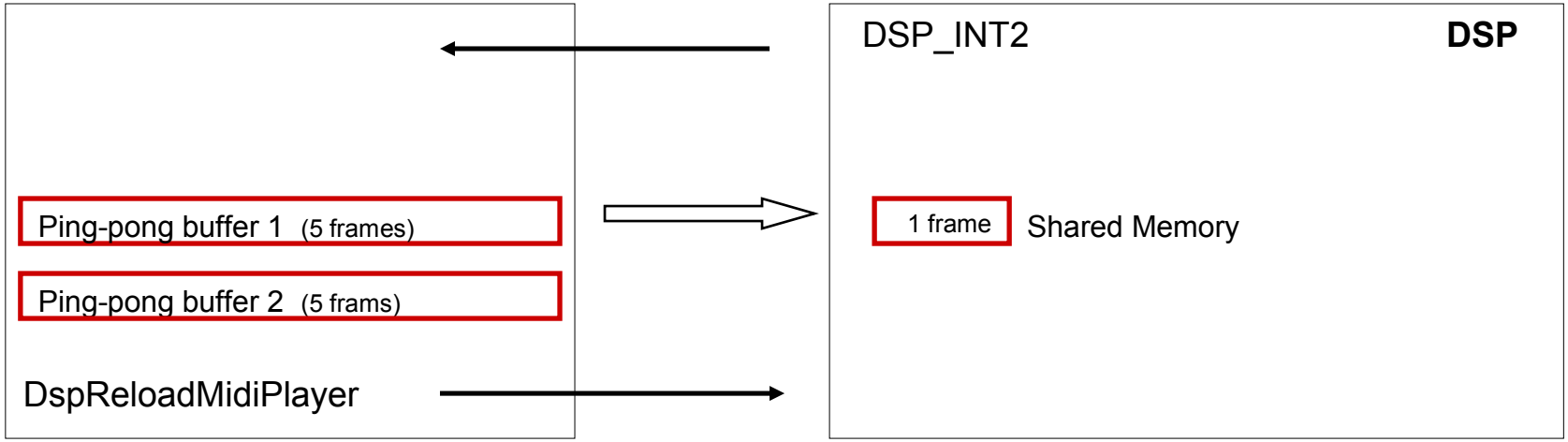
NEONSEVEN

Midi Driver – **reload**

When the midi module has played the current frame, it ask new midi data genrateing **DSP_INT2** Interrupts for the MCU.

The interrupt service routine **MIDI_Irq_proc()** copy a new frame (from ping-pong buffer) into Shared Memory and send command “DspReloadMidiPlayer”

MIDI_Irq_proc()

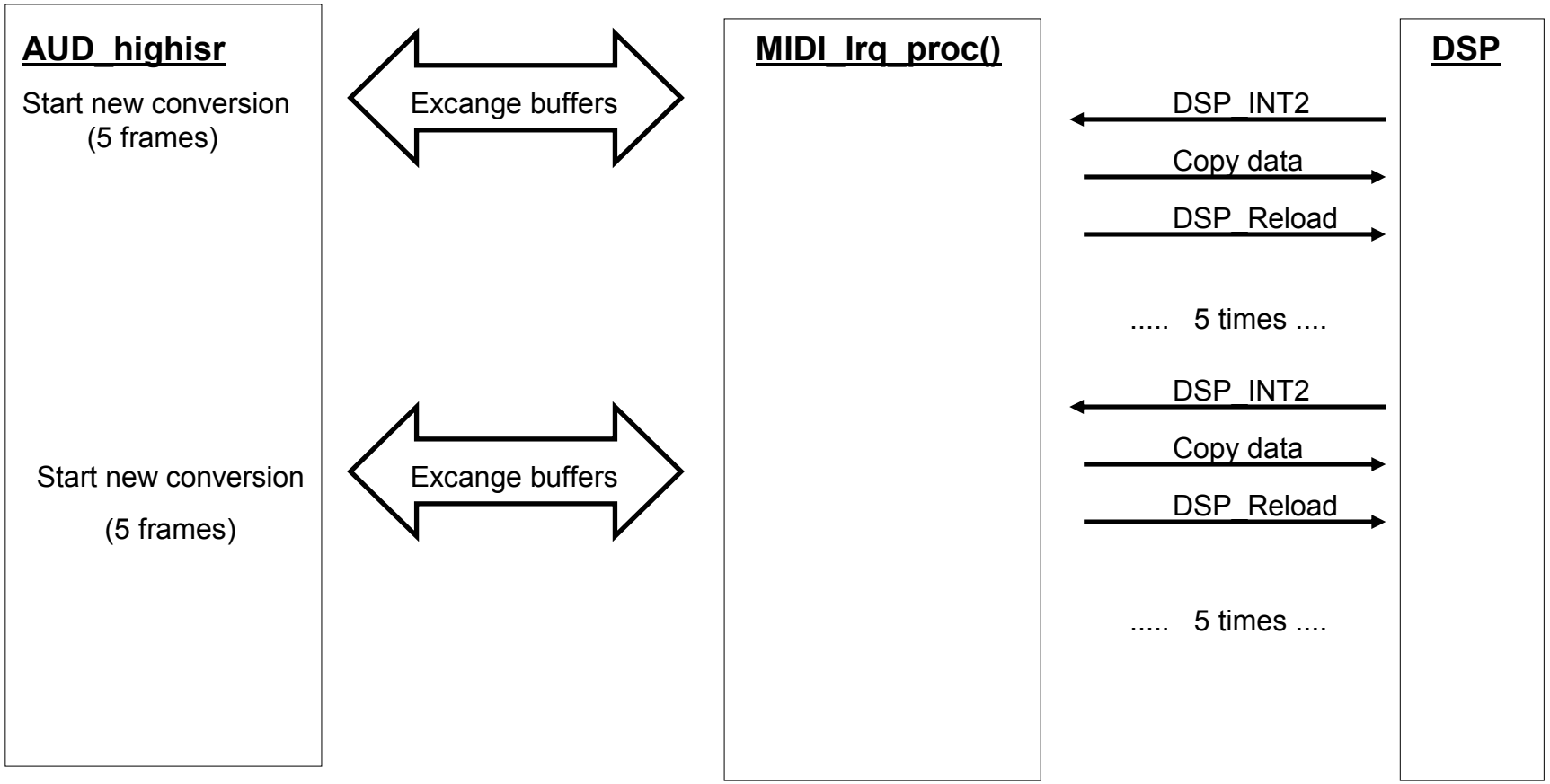


Midi Driver – **reload – Midi parser**

- The Midi parser has been implemented into the function **ConvertMidiFrame**. (with every call it convert only one frame).
- The conversion is a "heavy" operation so it cannot be performed into the interrupt service routine. On the contrary, it is performed by a lower priority task: OS_PROCESS(**AUD_highisr**)
- For this purpose two **ping-pong buffers** has been introduced. Each of these buffers memorizes 5 frames (5 arrays of 65 words). 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 file.
- The MIDI_Irq_Proc () every 100ms (5 frames), exchanges the two buffer and activates again the parser by setting the **audSemaPID**

Midi Driver – **reload**

- Every frame has **65 words** of midi data that is about 20msec of music, so the parser transalte 5 new frames every 100msec.



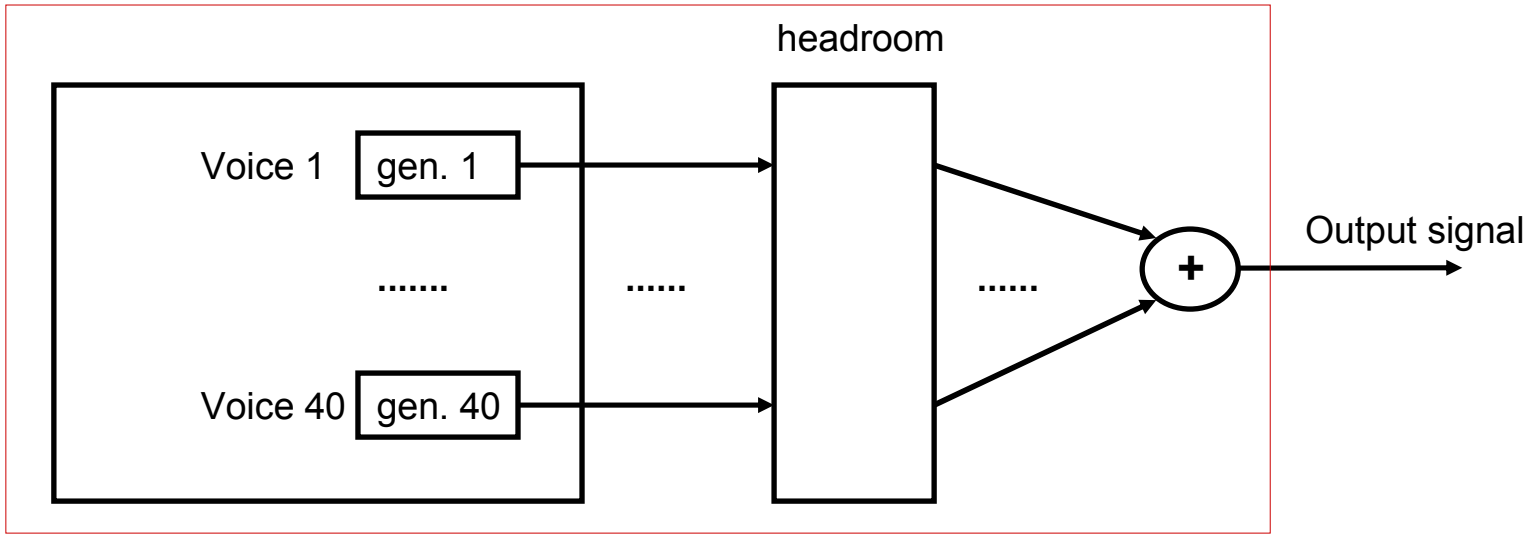
Midi Driver – **reload – status word**

With every interrupt the DSP return to the MCU three words:

➢ Status word

- **Bit 0 (Limit flag):** If this bit is set, Overflow (clipping) occured adding the output data of all voices in the last frame, so the headroom should be increased. (Headroom is the number of applied right shifts for each internal generator output)

Midi module



Midi Driver – **reload – status word**

- In the present version of midi driver the headroom is fixed to 3. And the clipping control is performed by the function "**NormalizeMidiVolume**".

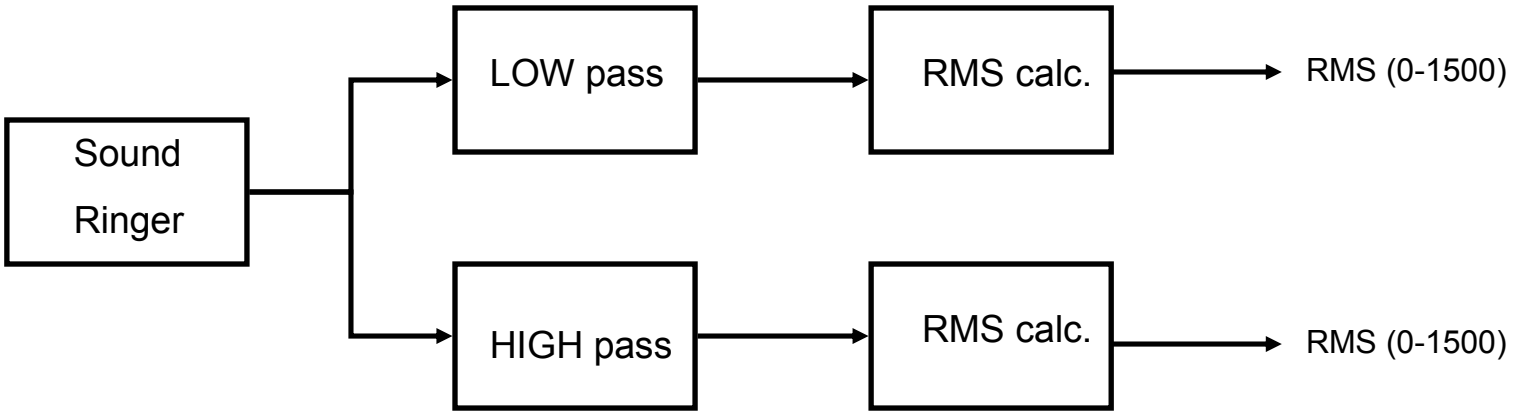
  It scan the whole midi file cecking the volume events, then it rewrite those volume controlls to avoid clipping.

- **Bit 1 (Metronom flag):** The Metronome Flag indicates that a music beat (by default ¼ Note beat,) has occurred in the last frame

Midi Driver – **reload – RMS**

➢ RMS low / RMS high words:

High and Low RMS are logarithmic RMS values calculated over 20ms of audio data (one frame). The maximum value is 1500 (corresponding to 0dB). The high and low channels are derived by applying a simple first order FIR low-/high pass filter before the RMS calculation.



❑ With every interrupt the tree words are read by the interrupt service routine **Midi_irq_proc()** and they are used to flah LEDs.

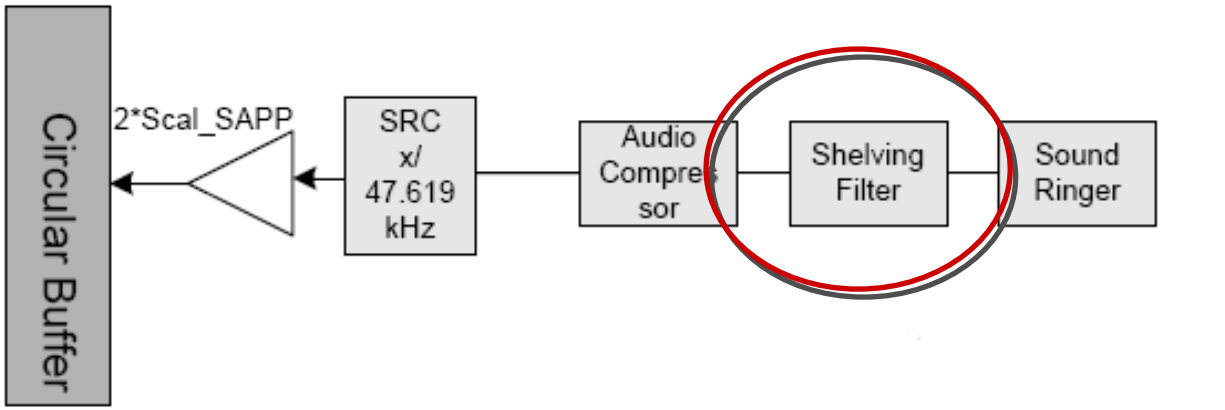
NEONSEVEN

Midi Driver – Shelving Filter

- This module is mainly **used for high frequency boost** in audio siognals.
- Its transfer function is given by:

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

- The filter is activate/disactivate by the DSP **AUDIOPOSTPROC command**.
- The same command can be used to change the filter coefficients (**b_exp**, **b0**, **b1** and **a1)**

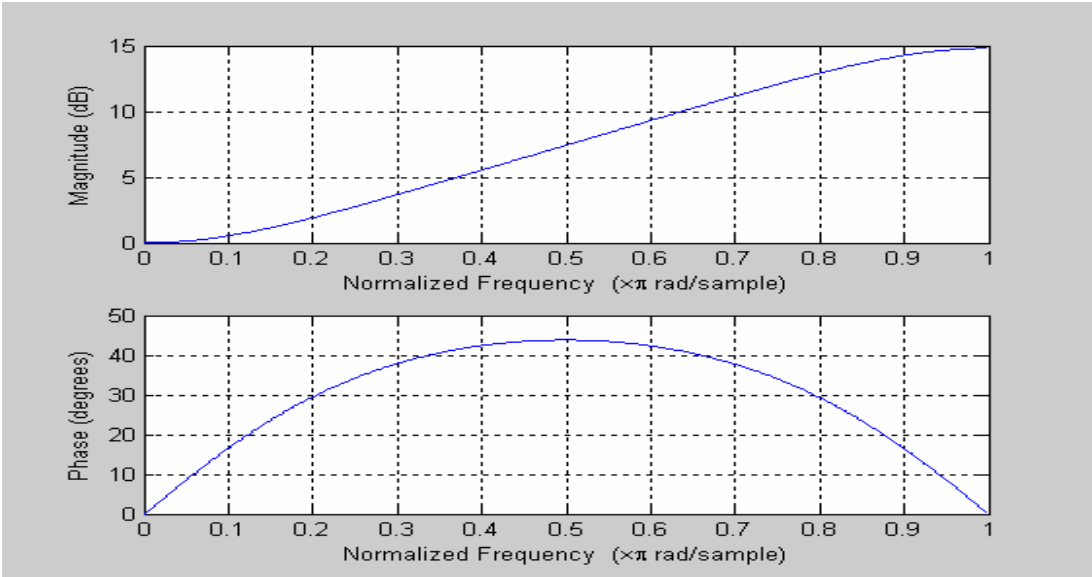


Midi Driver – Shelving Filter

- By default the filter coefficents are fixed to:

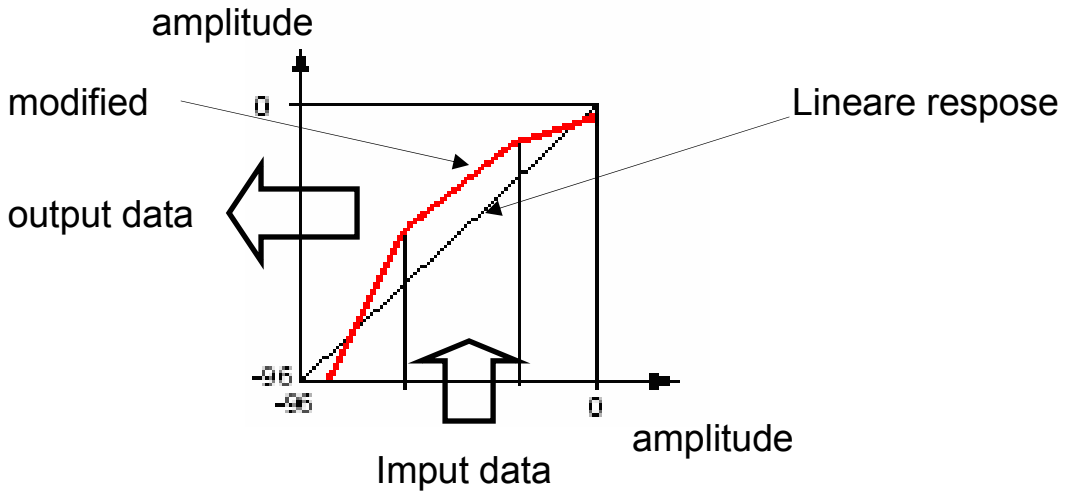
Parameter Name	Default Value
b_exp 2	2
b1	0xE19A
b0	0x4B33
a1	0x3333

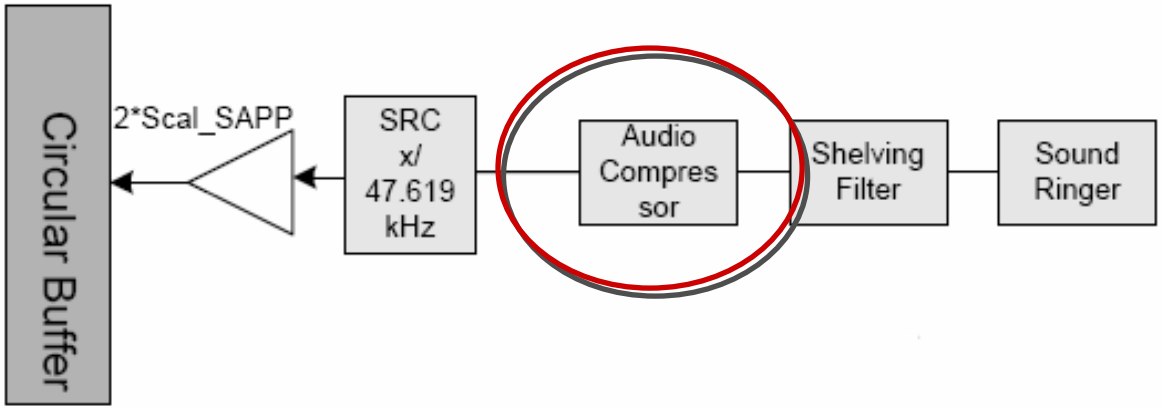
- The corresponding **Frequency Response**:



Midi Driver – Audio Compressor

- The audio compressor is a device **for manipulating the dynamic range** of audio signals:
- IDEA:





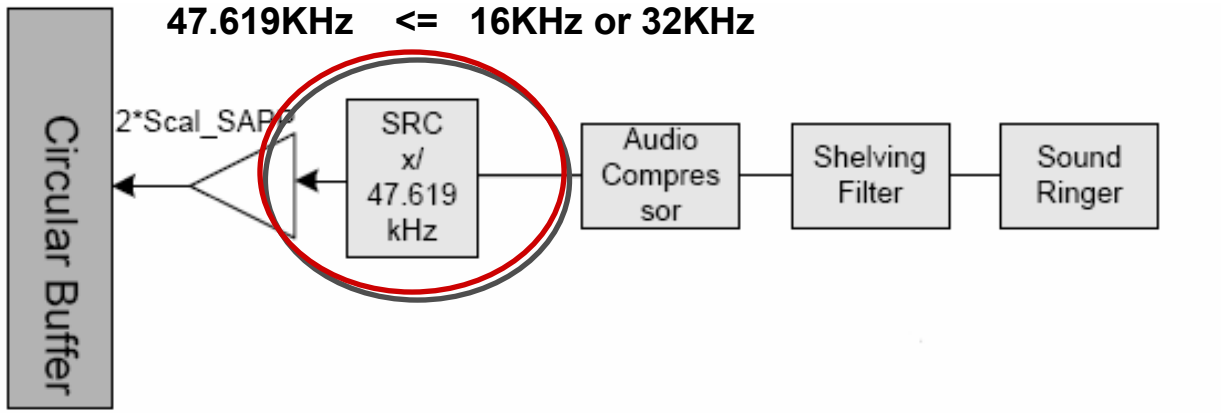
Midi Driver – Audio Compressor

- The Audio Compressor is activate/disactivate by the DSP **AUDIOPOSTPROC command.**
- the same command can be used to set the compressor coefficents

- In the present version of midi driver the audio compressor and the shelving filter are both activeted and they work with the default coefficents

Midi Driver – frequency converter

- The circular buffer is a 3000 word buffer to store and mix the voiceband samples from external sources.
- The signals to be stored should have a sampling rate of **47.619kHz**, therefore between the sound ringer and the buffer there is a frequency converter.



NEONSEVEN

Midi Driver - Files

- Aud_intf.c - Interface functions
- Aud_main.c - dispatcher
- Aud_ear.c- midi driver State Machine
- Aud_intern_midilib.c
- Aud_intern_midilib.h - Contains prototypes, enums, typedef etc.
- Aud_lib.c - Function library (DSPstartMidiPlayer; DSPstopMidiPlayer;...)
- Aud_tone.c - Definition of some midi files kept in flash

NEONSEVEN

SMF (standard midi file) VS. SP-MIDI

Channel: The midi data is organizzed in channels, every channel is associated to a different instrument and every midi file can send commans to 16 different channels.

Voices: is the max number of notes that the sinttesizer can play togheter

- The number of voices associated to a single channel changes run-time

NEONSEVEN

SMF (standard midi file) VS. SP-MIDI

There are two main differences between standard midi file and sp-midi

➢ Drum channel

- In the **Standard Midi File** only the channel 10 is dedicated to drum instruments
- In **SP-Midi** protocol there are two drum channel: 10 and 11. (By default only the channel 10 is set like drum channel)

➢ Cannnel priority

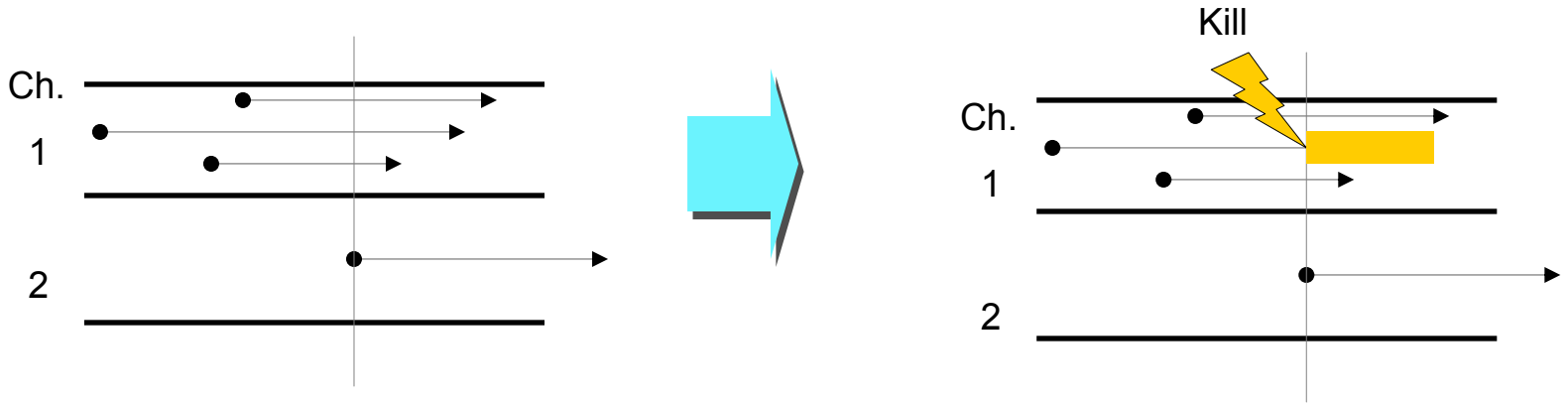
- In the **Standard Midi File** the channel priority is not defined
- with **SP-Midi** protocol it is possible to define a channel priority table.

NEONSEVEN

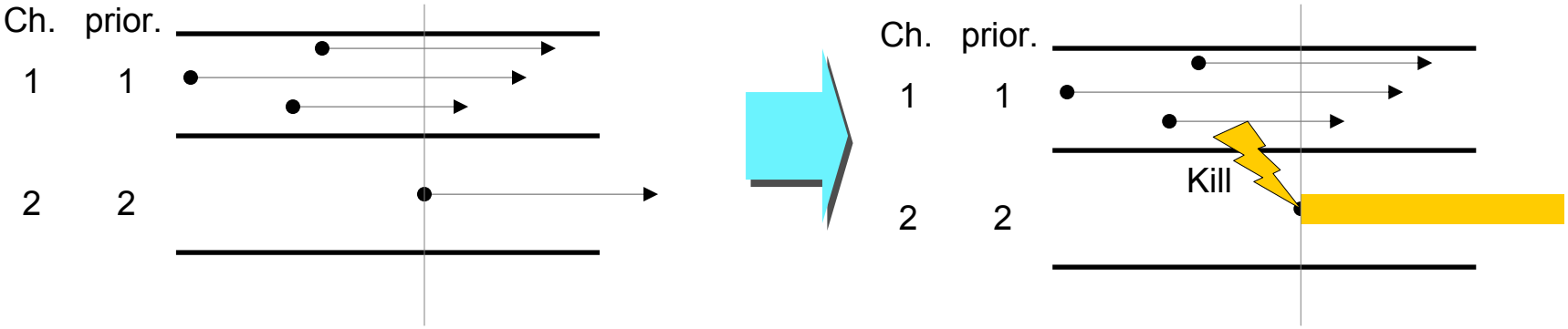
SMF (standard midi file) VS. SP-MIDI

Example: **2 channels**; **n of voices 3 (max)**

SMF



SP-Midi



NEONSEVEN

Reference documents

- ✓ EgoldLite Firmware Manual, Rev. 1.02
- ✓ Audio driver intrface specificatio, Rev. 0.9.22
- ✓ Audio Compressor: Description of the Configuration Paramiters, ver. 04
- ✓ Interface description for the High Frequency Shelving Filter, ver. 01
- ✓ Internal Midi Player Specification, Rev. 1.01