PWM Driver
Ver. 1.06

Specifications

The PWM Driver controls sound production through the communication port. It turns sound production on and off, changes timbres (selecting the output data), sets the volume, and specifies musical intervals by transmitting control data to the communication port. The transmission of control codes is always treated as the transmission of realtime data. Therefore, as in the case of the MIDI, controls must be provided from the Sound Driver to ensure synchronization. As in the case of an FM sound source chip, the PWM Driver only has the function of producing a sound in response to a given piece of data. Notice that the PWM Driver by itself cannot play a piece of music on the basis of a sound sequence.

Although the CPU for the PWM Driver uses an SH2 master device, when the PWM Driver is to be incorporated into a game, the master device can be changed into a slave device.

The PWM Driver outputs from four channels by means of a waveform synthesis. Because it needs to control four channels, the PWM Driver makes exclusive use of the communication port in 4-word sounds, at the four specific addresses listed below. The communication port should not be used for any purpose other than sound production.

```
0x20004028          channel 1
0x2000402A          channel 2
0x2000402C          channel 3
0x2000402E          channel 4
```

The PWM Driver can be assigned to any address. In the following, the starting address of the PWM Driver is assumed to be XXXX.

The data to be output to a port is in the following format, defined in units of words:

High byte: Volume. The high 4 bits indicate the left volume, the low 4 bits the right volume. For both right and left volumes, an 0 indicates the minimum volume, and a 15 the maximum. The right volume is used in the case of the Qsound.
Low byte: Timbre number. Values 0 through 254 can be set in this byte.

Timbre files that are created through the use of Tone Editor 32X require four long words (16 bytes) per timbre as a header (PWM data area) from the beginning of the file, in the following format:

Timbre number n:

.data.1	voice address	Data storage address for timbre number 0. Relative value from XXXX.
.data.1	data size	Data size of timbre number n.
.data.1	loop point	Loop-point relative address. A loop is executed only when this address is 0.
.data.1	address up counter	Address-up counter.

"Address-up counter" refers to the data that is used to read sampled data strings on a phase-by-phase basis. An address-up counter is analogous to the pinch roller in a tape deck or to the frequency. For details, see "7. PWM (for Super 32X)" in the "Tone Editor 32X" manual.

PWM Sound Sources for the 32X

A PWM sound source has voltage values that are stored in capacitors as immediate data. Although in this respect a PWM sound source differs from a PCM sound source, both PWM and PCM sound sources share the same attribute in that they store data as waveforms. Therefore, for the purpose of developing a sound program, the difference between PWM and PCM sound sources is immaterial. However, because the 32X needs to send data by software for each sampling operation, it entails a considerable CPU overhead. Thus, when producing sounds from the data that is sampled at 44100 Hz, which is the sampling frequency used in a CD, the 32X needs to write data 44,100 times. Measuring this timing simply by means of interrupts requires a precision 735 times greater than that required in the case of a V-INT.

The CPU always accesses a PWM sound source through the FIFO. The data stored in the FIFO is automatically transmitted word-by-word at the interval that is set in the cycle counter. While producing sounds, the sound program needs to keep the FIFO supplied with data so that the FIFO does not become empty.

The timing at which data is written to the FIFO can be determined by checking the FULL bits of pulse width registers (0x20004034, 0x20004036, and 0x20004038). When a PWM interrupt is used, the data-writing timing can be synchronized with the "FIFO to PWM" data transfer timing by setting the value "PWM timer interrupt interval = 1" to the PWM control register (0x20004030). However, because a perfect synchronization cannot be achieved, the user needs to check to see whether or not the FIFO is empty.

In actuality, a conversion from PCM data to PWM data is performed by means of an MSB inversion (offset-binary conversion). However, because PWM does not use negative values and 0, which is PCM's central value, is replaced by 0x80, for a waveform synthesis operation the 0x80 value must be output during a silent state. This means that when multiple processing tasks are

carried out, in which a waveform synthesis is performed on a case-by-case basis, a great variability in sound volume can result.

If a waveform synthesis is not required, the volume should be increased gradually from the 0 level in order to reduce the noise and to provide a long enough charge time to the capacitors. Likewise, at the end of data the volume should be decreased gradually toward the 0 level.

Initialization

An initialization is required, but it should be performed only once, as follows:

```
mov.l     #h'xxxx+4,r0
jsr       @r0
nop
```

When control is returned from the initialization routine, the contents of registers r0, r1, gbr, and sr are destroyed.

The initialization routine only enables the PWM interrupt. If this presents a problem, the user should write the user's own initialization routine by referencing the source code.

PWM Interrupt-Routine Call

In the PWM interrupt, always issue the call indicated below. A call destroys the contents of registers r0 and r1. In the code indicated below, registers r0 and r1 are not saved for conformance to the multi-interrupt processing routine that is described in the "Technical Information" document.

```
mov.l     #h'xxxx,r0
jsr       @r0
nop
```

Setting a Waveform Address

When started, the PWM Driver does not recognize the address at which waveform data is stored. Therefore, after completing the initialization, set a waveform address as follows:

1. Write 0xFFFF to 0x20004028.
2. Write the high word of the waveform address, viewed from SH2, to 0xFFFF to 0x2000402C.
3. Write the low word of the waveform address, viewed from SH2, to 0xFFFF to 0x2000402E.
4. Write 0xFFFF to 0x2000402A.

Using the Qsound

The Qsound can be started by writing the value 0x01FF to address 0x20004028. Note that when the Qsound is used, only channel 1 is enabled; channels 2 through 4 remain idle.

To fully exploit the capabilities of the Qsound, the user needs to supply pan position data on a realtime basis. There are 31 pan positions, 0 being the leftmost position and 31 being the rightmost position. Therefore, the center position is designated as position 15. A pan position can be implemented by writing the following data to address 0x20004028:

Location	Data
0	0x80FF
1	0x81FF
2	0x82FF
3	0x83FF
4	0x84FF
30	0x9EFF

To reset the Qsound, write 0x02FF to address 0x20004028.

Stopping a Sound

To turn off a sound, write 0x00FF to the dedicated port for each channel. To turn off a sound for all channels, code as follows:

```
mov     #0,r0
mov.l   #h'20004020,r1
mov.w   r0,@(h'08,r1)
mov.w   r0,@(h'0A,r1)
mov.w   r0,@(h'0C,r1)
mov.w   r0,@(h'0E,r1)
```

Reassembling

As supplied, the PWM Driver cannot be incorporated into a game in its binary form. Before the PWM Driver can be incorporated, it needs to be assembled by resetting the value XXXX at the source level. To do so, replace the symbol "Pwmint" in the file "assign.i" with XXXX, and then assemble the file "pwmdrv.src".

Information

When the sound production mode is off, the cycle counter does not operate, and no data can be written to the FIFO. However, when the power production mode is turned on for the first time after the system is powered

up or reset, undefined data flows from the FIFO and causes a noise (BUCHI noise [BUCHI - UNKNOWN TERM]). There is no way to deal with this noise.

A writing operation which is performed when the FULL bit in a pulse width register is on causes the FULL bit to be turned off. This problem should be resolved in software.

If the sound production mode is turned off when the FULL bit in a pulse width register is on, and if subsequently the sound production mode is turned on, both FULL and EMPTY bits are turned on. This problem should also be resolved in software.

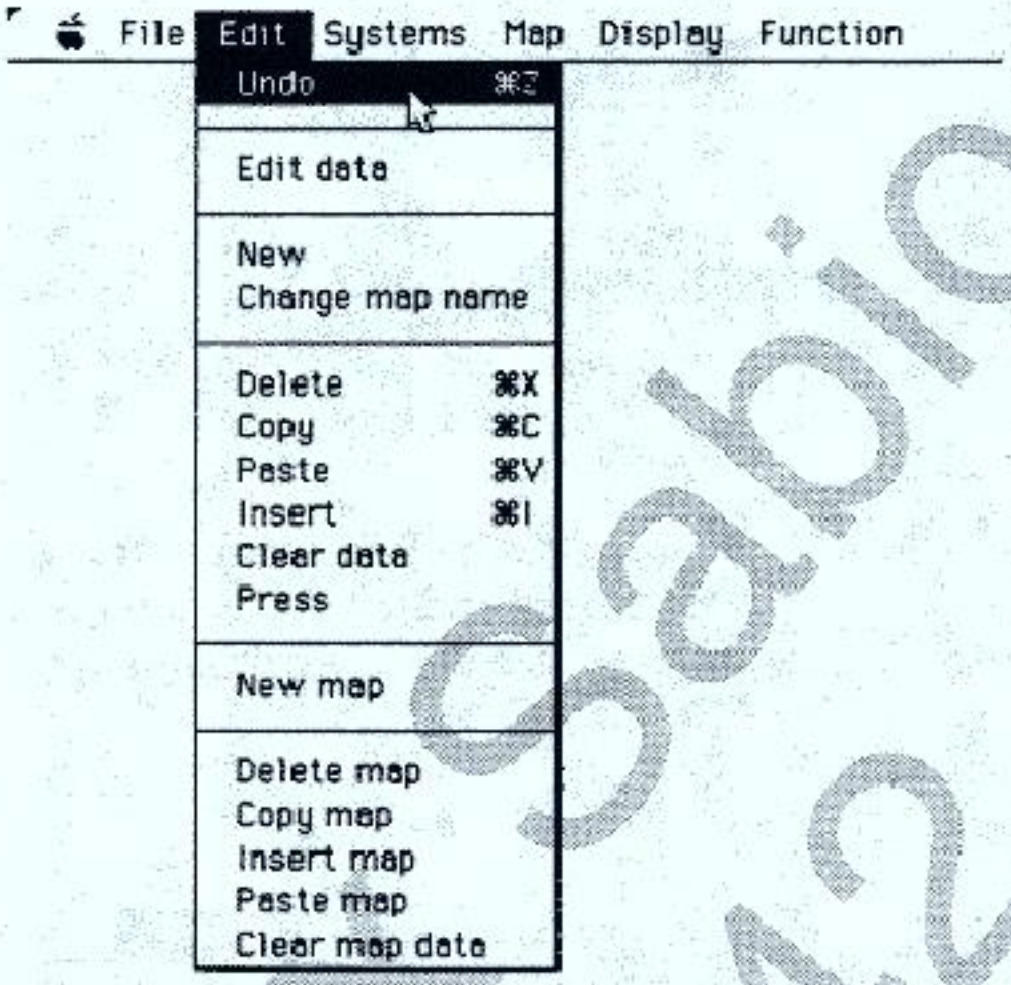
Changes in Version Upgrade

Tone Editor 32X Ver. 1.01 (March 12, 1995)

The actual Tone Editor 32X software is Ver-1.01, whereas the manual that is supplied in conjunction with the software is Ver-1.00. The difference in version number is due to corrections that were made to the bugs found in Ver-1.00. Ver-1.01, however, is identical to Ver-1.01 in operation and functionality.

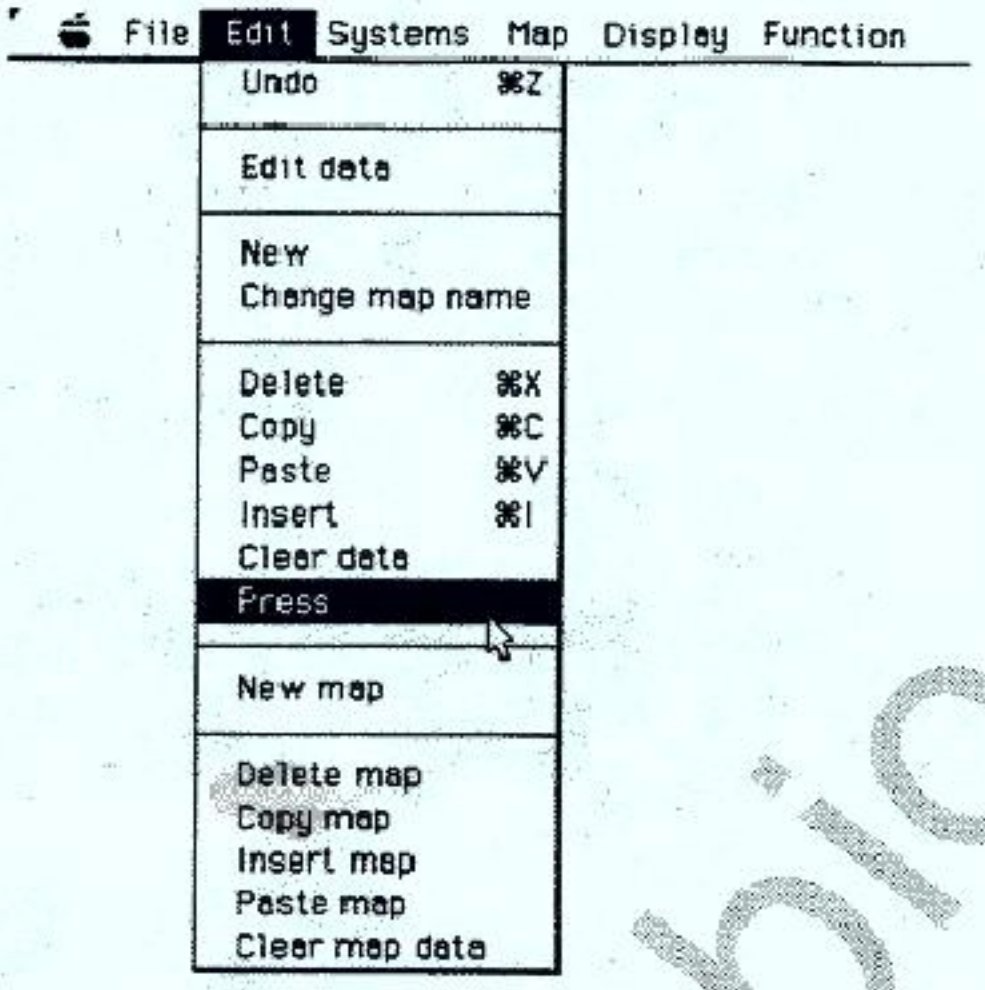
32X Sound Simulator Ver.1.10 (March 10, 1995)

Undo



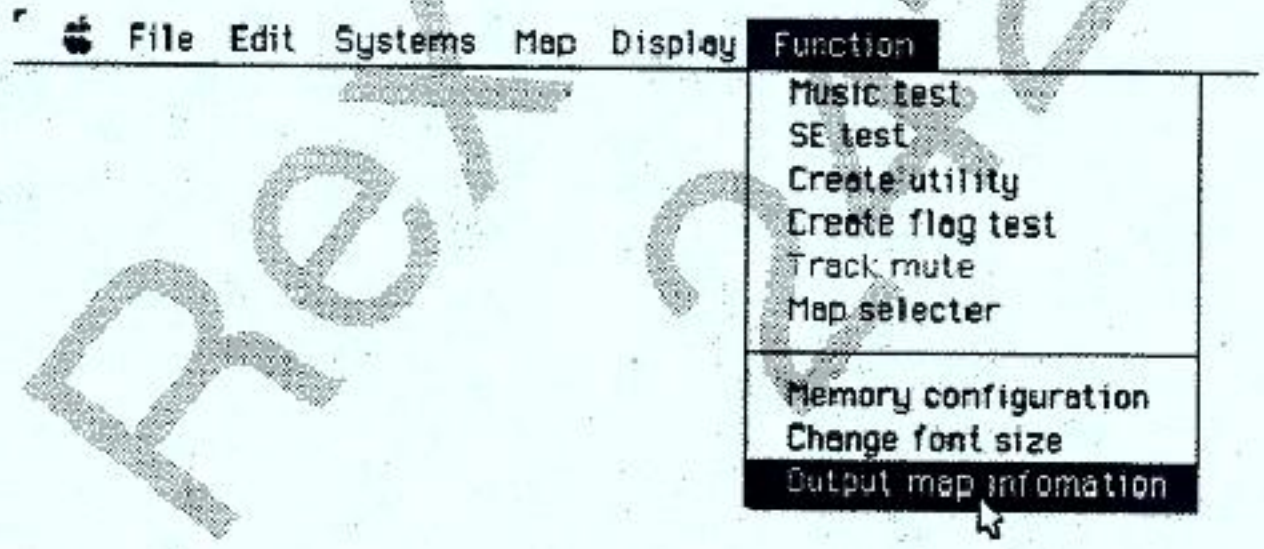
The Undo function is now available for use on the Cut, Paste, Copy, and Delete functions on the Edit Window.

Press



The Press function has been added to the Edit menu. This function fills any gaps that are created by deleted blocks and can be used when it is necessary to adjust the spacing between blocks after the [Delete only] option is selected in the Delete function of the Edit menu.

Output map information



The Output Map Information function saves the contents of the Edit window as a text file.