Vircon32

32-BIT VIRTUAL CONSOLE

System specification

Part 5: The sound chip (SPU)

Document date 2023.01.08 Written by Carra

What is this?

This document is part number 5 of the Vircon32 system specification. This series of documents defines the Vircon32 system, and provides a full specification describing its features and behavior in detail.

The main goal of this specification is to define a standard for what a Vircon32 system is, and how a gaming system needs to be implemented in order to be considered compliant. Also, since Vircon32 is a virtual system, an important second goal of these documents is to provide anyone with the knowledge to create their own Vircon32 implementations.

About Vircon32

The Vircon32 project was created independently by Carra. The Vircon32 system and its associated materials (including documents, software, source code, art and any other related elements) are owned by the original author.

Vircon32 is a free, open source project in an effort to promote that anyone can play the console and create software for it. For more detailed information on this, read the license texts included in each of the available software.

About this document

Summary

Part 5 of the specification defines the console's sound chip (SPU). This document will describe the behavior of this chip, its control ports, its provided sound commands and the process it uses to produce audio output.

1 Introduction

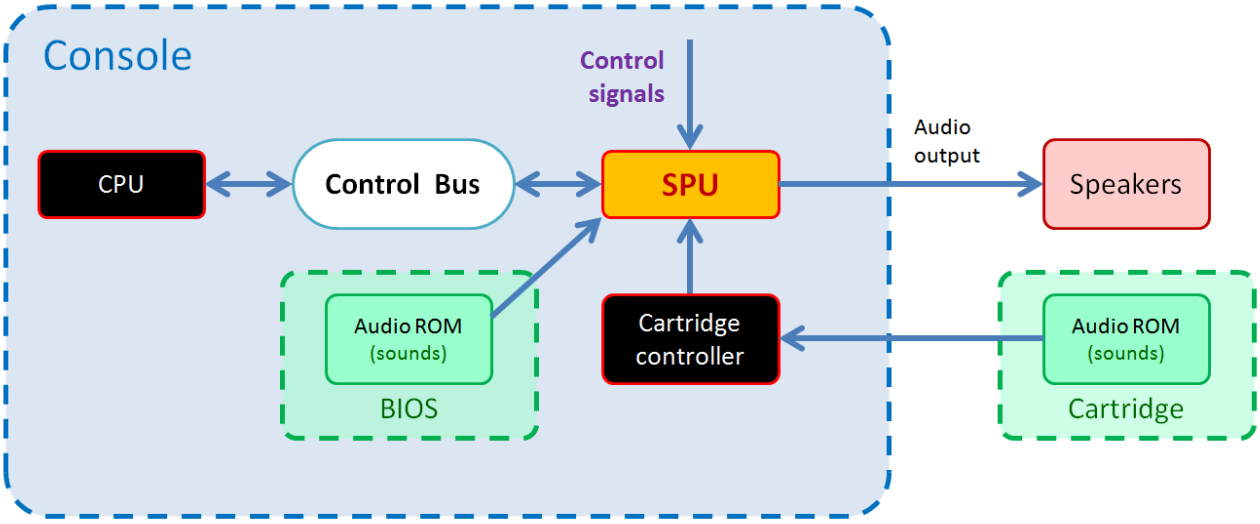
The SPU is the audio chip in the Vircon32 console. It is responsible for creating all sound that can be heard through the speakers. To do this it will play the sounds contained in both the BIOS and the cartridge and apply basic audio effects to them.

This sound chip is quite simple: it features an array of 16 sound channels, that can each play one of the available sounds at any given moment. To produce output, the sound from all currently playing channels is then mixed together.

The activity of sound channels is controlled via sound commands by the processor. Aside from this there are a few audio effects that can be applied to channels' playback. This will be covered in later sections.

2 External connections

Since the SPU is just one of the chips forming the console, it cannot operate in isolation. This figure shows all communications of the SPU with other components. As shown, the SPU has connections to all available audio ROMs to be able to use their contained sounds for playback. Note that, for clarity, the console diagrams in part 2 of the specification purposely omitted these connections.



Each of these connections will be explained individually in the sections below.

2.1 Control signals

As all console components, the SPU receives the signals for reset, new frame and new cycle. The responses to those signals are detailed in section 10 of this document.

2.2 Control Bus

The SPU is connected as slave device to the Control Bus, with device ID = 3. This allows the bus master (the CPU) to request read or write operations on the control ports exposed by the SPU. The list of SPU ports as well as their properties will be detailed in later sections.

2.3 BIOS chip

The BIOS, which is always present, contains an audio ROM with exactly 1 sound. The SPU can access that sound through its sound slot with ID = –1.

2.4 Cartridge controller

Same as with the BIOS, the SPU can access any sounds present in the cartridge through sound slots with IDs 0 to 1023. Note however that the connection must use the Cartridge Controller as a proxy, since there may not be a cartridge present. And even when a cartridge is connected, each of them can contain a different number of sounds (0 to 1024).

2.5 Speakers

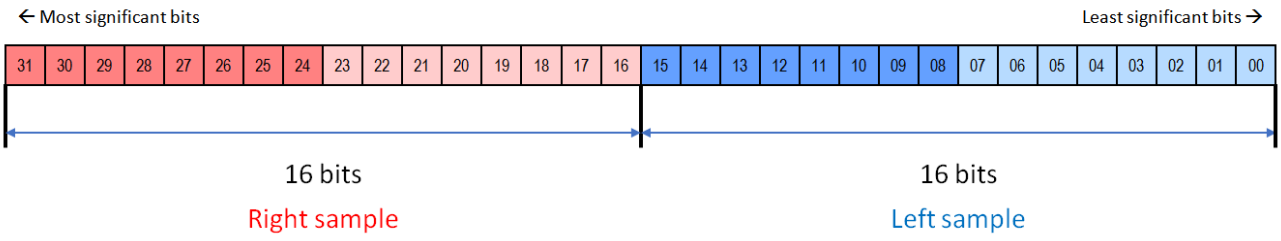
The SPU needs to be able to generate sound for all playing channels, mix it and send the result to the speakers at the correct playback frequency so that it can be heard. This is done via the audio output connection. Section 9 will cover how this output is done.

3 Working concepts

Before explaining the SPU's sound functions or the internal variables that affect them, we must present a series of basic concepts that the SPU is built around.

3.1 Sound samples

The minimal unit of audio information handled by the SPU is the sample, which represents a single discrete value in the sound sequence for either the speakers or a sound. Every sample is represented as a single 32-bit value that encodes 16-bit values for left and right channels, as already documented in part 2 of the specification:



3.2 SPU sounds

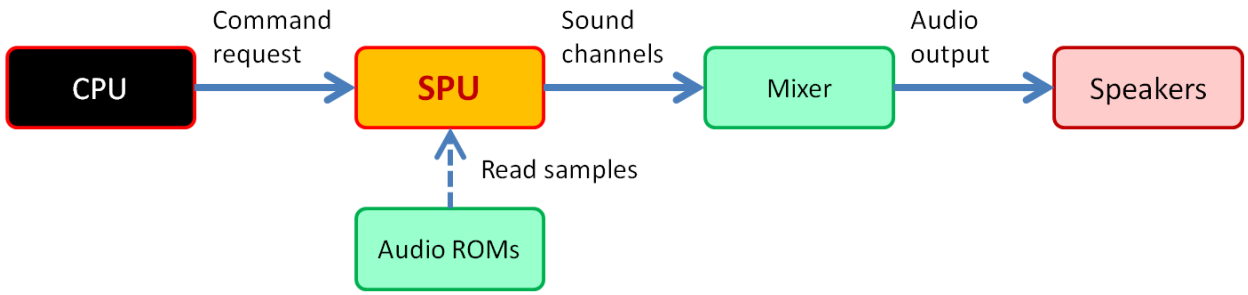
An SPU sound is a sequence of sound samples, stored with the same format and playback rate as the console's audio output signal, this is: stereo, 16-bit samples played at a rate of 44100 samples per second. Sounds can be of any length from 1 sample up to the whole size of its containing audio ROM. Samples within a sound are numbered with positions beginning at 0. All SPU playback functions need to use a sound to generate audio.

The SPU identifies and accesses its available sounds through an array of numbered sound slots. There are 1024 slots usable by the cartridge (with sound IDs from 0 to 1023), and an additional slot with ID = –1 for the BIOS sound.

3.3 Sound channels

A sound channel is a basic audio generator that can be assigned an SPU sound and use its samples to create an output playback. Sound channels support the basic playback control commands (play, pause/resume and stop). The SPU features an array of 16 sound channels, and each of them is identified with a numeric ID from 0 to 15.

Sound channels use a playback format identical to the console's audio output signal. That allows the SPU to mix together the output of all currently playing channels into a single output. For every new frame, the SPU will generate the appropriate number of samples for a frame's time and store them in a buffer. From there the SPU will continuously send the sequence of output audio samples to the speakers with the correct timing. That will make all sounds being played in sound channels audible. As a whole, the operation of the SPU with sound channels looks like this:



3.4 Connection to audio ROMs

The SPU does not contain any sounds itself, so it needs to read the sounds stored in the BIOS and cartridges by connecting to their respective audio ROMs. An audio ROM is a read-only memory region that contains a sequence of sounds. Each of these sounds can have any possible length, from 1 sample up to the whole audio ROM. Audio ROM sizes have size limits: The SPU can only handle audio ROMs of sizes up to 256 x 1024 x 1024 samples for cartridges, and up to 1024 x 1024 samples for BIOS.

When a cartridge containing N sounds is connected, the first N slots from ID = 0 become assigned to each of the sounds, in order. The rest of slots up to ID = 1023 will become unused and not accessible. The slot with ID = -1 gets assigned to the BIOS sound, which is guaranteed to exist and be unique.

This connection process happens whenever a new cartridge is inserted. Still, in Vircon32 systems cartridges can only be inserted while the console is off. Therefore it is safe for implementations to delay any needed procedures for the SPU to establish this connection until the next console power on.

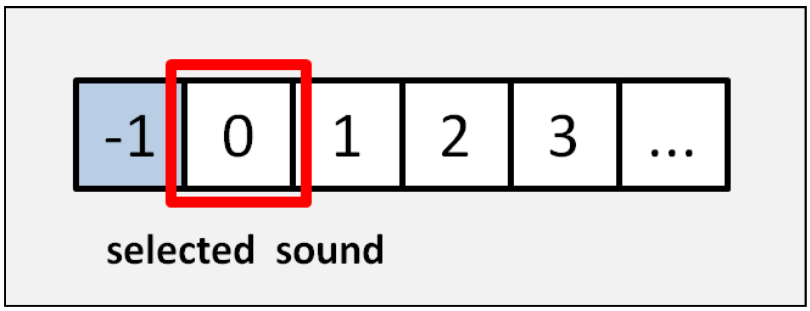
It is up to the implementation to decide how to establish a connection to audio ROMs and locate and read their sounds. For example, it is possible to read all sounds in audio ROMs beforehand upon connection. Another option would be to keep pointers to sounds and have the SPU will read sample values on the fly.

3.5 Selected elements

The SPU organizes its audio elements (sounds and channels) in sets. Sound generation can use all channels and sounds it needs at once, but other SPU functions will typically operate with only one element from each set at any given time.

Selected sound

All assigned sound slots form a continuous range of usable sound IDs as shown in this image. To determine which sound to use when executing sound commands or applying internal variables, the SPU always considers one of those IDs as "selected". The sound selected by default is the BIOS sound, since it is the only one guaranteed to always exist.



Selected channel

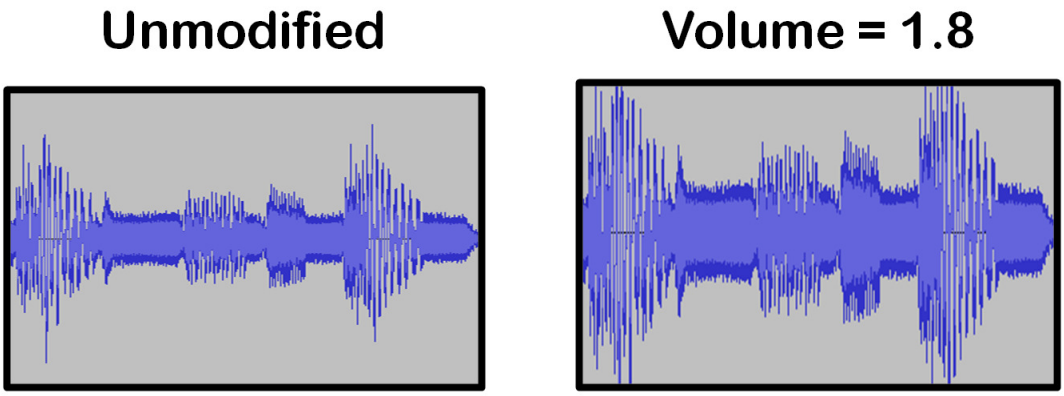
Similarly, available sound channels form a range of usable channel IDs from 0 to 15. To determine which channel to use when executing sound commands or applying internal variables, the SPU always considers one of those IDs as "selected". The channel selected by default is the first, corresponding to channel ID = 0.

4 Audio effects

To increase flexibility and achieve more advanced sound features, different types of effects can be applied to configure each SPU sound channel. These effects, when enabled, will modify the way a channel plays its assigned sound and therefore change the result heard through the speakers.

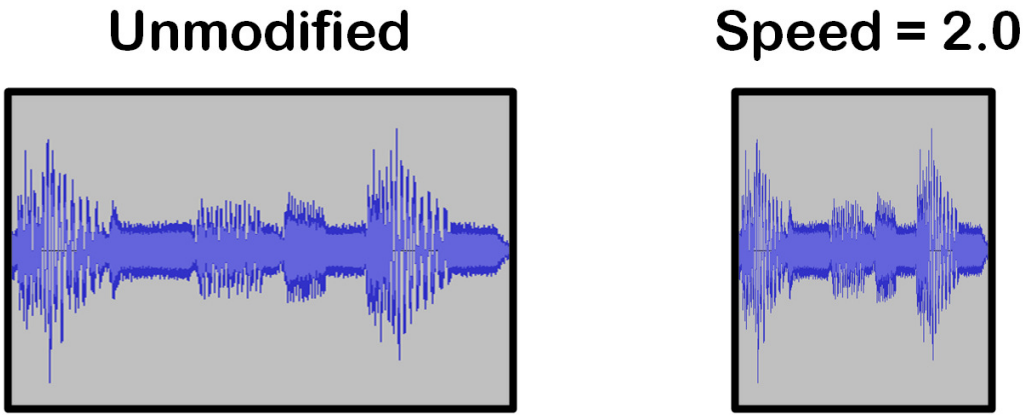
4.1 Volume effect

When playing a sound, an SPU channel can be configured to modify the playback volume. The channel's volume parameter acts as a linear scaling factor. As a result the output sound wave gets scaled accordingly. An example of this effect could look like this:



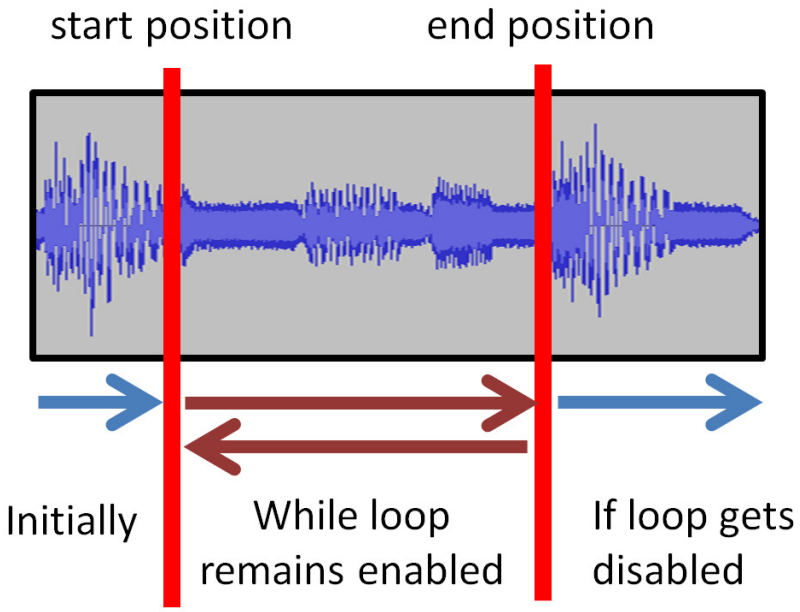
4.2 Speed variation effect

Another configurable parameter featured in SPU channels is their playback speed. A channel's speed parameter will be used to determine the rate of advance along its assigned sound's samples, resulting in a change in both speed and pitch. An example of modified playback speed would look like this:



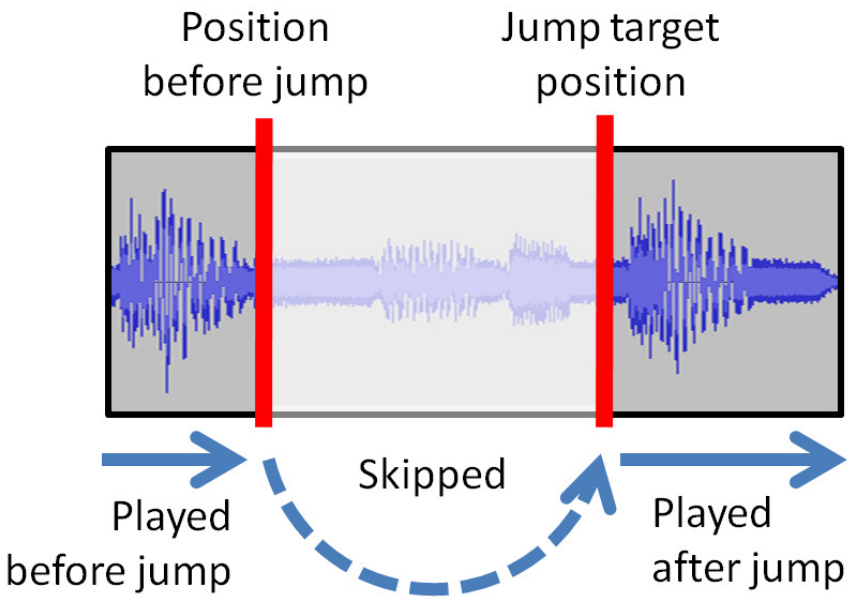
4.3 Looped playback

When enabled, this effect causes a channel to continuously loop over a defined region of its assigned sound. The start and end limits for the looped region are configurable for each SPU sound. Playback of a sound with loop enabled can have 3 stages as shown here:



4.4 Playback jumps

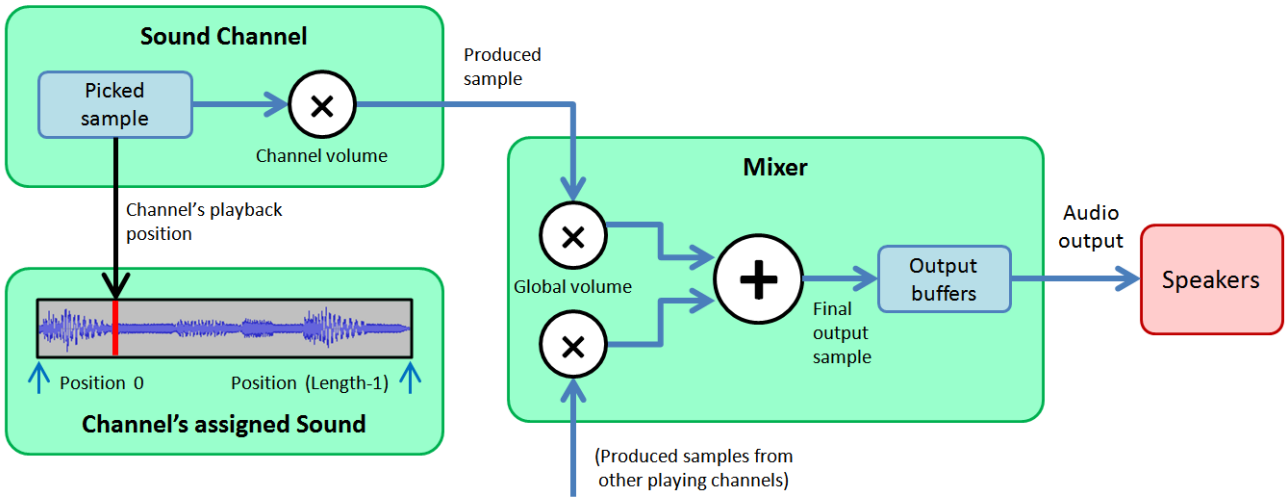
Channels play their assigned sound by keeping a playback position that continually advances along the sound. Channels allow this playback position to be modified by program commands, resulting in jumps. For example, in the case of a jump forward, playback of a sound would happen as shown in the image below.



5 Sound generation

Sound generation is the process used to produce new output audio samples from the existing source SPU sounds. To achieve this, the SPU first produces individual output samples from each currently playing channel, as determined by their configuration. Those channel samples are then mixed into a single, global output sample. Finally, a sequence of mixed output samples is stored and delivered to the speakers via the audio signal at the timing needed by the playback rate of 44100 Hz.

The global sound generation process can be summarized in this diagram:



5.1 General process and timing

The sound generation process is triggered at the beginning of each frame. For every execution this process will produce the needed output samples for 1 frame, this is, an ordered sequence of 44100 / 60 = 735 samples. To achieve this, the following steps are repeated 735 times:

1. Generate the current sample for each sound channel currently playing.
2. Apply the mixing logic to join all channel samples into a single output sample.
3. Store the output sample in an output buffer, ready to be sent for playback.
4. Advance all currently playing channels.

Each of these steps is detailed in the sections below.

An important consequence of generating sound only at the beginning of frames is that the state of sounds and channels between those instants is not relevant for sound generation. For instance: let's say that within a frame a program sets channel 2 to volume 0.5. Later in that same frame, channel 2's volume is changed to 0.8. In that case:

- If sound generation was continuous, channel 2 would have generated some output samples with volume 0.5, and later other samples with volume 0.8.

- However, since sound is generated only between frames, channel 2 will generate all samples for that frame using the final volume of 0.8.

Similarly, if a channel is set to play and later stopped in the same frame, it will generate no sound at all on that frame because at the next new frame it will be stopped.

5.2 Generating channels output samples

Channels that are currently in playing state will simply pick from their assigned sound the sample corresponding to their current playback position. That sample will be multiplied by the present value of the channel's playback volume and the resulting value will be taken as the channel's output sample.

Interpolation methods

Since playback speeds can vary, channel playback positions will not always be integers. This forces implementations to decide on some logic to decide how channels should choose their picked sample.

The reference sound interpolation method in Vircon32 SPU is nearest neighbour. However, implementations are free to choose other interpolation methods for non-integer playback positions. Any method is acceptable as long as it does not distort the original sounds. This means that audio interpolations based on the original waveform, such as linear or cubic, can be considered compliant with Vircon32.

5.3 Mixing channel samples

The outputs produced by all currently playing channels will first be multiplied by the SPU's global volume factor. Then the resulting values will all be added. This result will be taken as the final output sample.

It should be noticed that the order of operations will affect the result. The order stated here (adding AFTER multiplying) is chosen to prevent saturation of the audio signal when several channels are playing at once and global volume needs to modulate them down.

5.4 Storing the mixed output samples

To ensure the correct timing of sound generation and output, some type of storage buffer will be needed. Its function is to hold generated output samples until it is time for the audio output signal to send each individual sample to the speakers. Possible strategies for this will be explored in section 9.

5.5 Advancing channels after each sample

After its current output sample was produced, a channel needs its playback position to be advanced. If playback speed was not configurable, channel position would advance 1 sample for every output sample (same playback speed). When we include a playback speed that acts as a multiplying factor, the advance after each output sample is simply:

Channel's playback position += Channel's playback speed

Loop processing

After the advance, the channel has to apply the looping logic. This will only be done if channel loop is enabled, and only if the sound loop configuration is valid. For a valid configuration a sound's loop end position must be greater than its loop start position.

Looping logic first needs to determine if the last advance caused playback position to go past the assigned sound's current loop end position. When that happened, playback position will "wrap around" to be set back into the loop. But, since playback speeds can be much greater than 1, this process needs to take into account that:

- Loop end could have been overshot by a value greater than the loop length itself.
- The correct new position for a seamless loop may not be the loop start.

In general, to ensure a correct loop, the new playback position needs to be set as follows:

1) Loop length = Loop end - Loop start +1
2) Full loops overshot = floor((Playback position - Loop end) / Loop length)
3) Partial loop overshot = Loop position - (Loop length * Full loops overshot)
4) Playback position = Loop start + Partial loop overshot

Finishing playback

After processing loop if required, the channel must check if playback position is greater than its assigned sound's last sample position. In that case the sound playback has been finished: the channel will be stopped and it will not generate any more samples for the present frame. If an implementation needs to always generate 735 samples for a channel, it may do so by filling the remaining samples with zeroes.

6 Internal variables

The SPU features a set of variables that store different aspects of its internal state. These variables are each stored as a 32-bit value, and they are all interpreted using the same data formats (integer, float, etc) described in part 2 of the specification. The only exception to this is the "Channel position" variable, as discussed below. Here we will list and detail all internal variables, organized into sections.

6.1 Global SPU variables

Global volume	**Initial value:** 1.0
Format: Float	**Valid range:** From 0.0 to 2.0

It represents the current output volume level of the SPU. Its value acts as a linear multiplier that will be applied by the mixer to its received samples from all channels. So, for each channel, the total volume multiplier will be (Global Volume * Channel Volume).

The most common use of the global volume is to avoid saturation of audio when mixing several channels, by using values smaller than 1.

6.2 Selected elements

Selected sound	**Initial value:** -1
Format: Integer	**Valid range:** From -1 to 1023 (*)

This value is the numerical ID of the currently selected SPU sound. The selected sound is the one that will be used in all sound related operations. It is also the sound affected by any sound configuration changes.

(*) The upper limit is given by the currently connected cartridge. If no cartridge is present, then the BIOS sound (ID = -1) is the only selectable sound.

Selected channel	**Initial value:** 0
Format: Integer	**Valid range:** From 0 to 15

This value is the numerical ID of the currently selected SPU sound channel. The selected channel is the one that will be used in all single-channel sound commands. It is also the channel affected by any channel configuration changes.

6.3 Configuration of each sound

The variables listed here are special. The SPU stores a copy of each of these variables for each existing SPU sound. This means there can be as many as (1024+1) copies. The implementation may decide if all these are always stored, keeping the ones for unused sound slots inaccessible, or if only the needed ones are created each time a new cartridge is inserted.

Together, each set of these variables describes the current configuration for a single SPU sound (see section 3 for an overview of how SPU sounds are defined). Only one set of these variables is accessible at any time, so each variable in this section is accessed by control ports via a “pointer” proxy. When the selected sound changes, those ports are all redirected to the copy of the variables for the correct sound.

Sound length	**Initial value:** (*)
Format: Integer	**Valid range:** From 1 to 268435456 (= 256*1024*1024)

It represents the number of samples of its associated sound.

(*): Its initial value is already set to the sound length in samples when the cartridge is connected or the console starts up. It will never change until the cartridge is removed.

Sound play with loop	**Initial value:** False
Format: Boolean	**Valid range:** True/False

When this parameter is true, its associated sound is configured to be played with loop enabled. When a channel receives a play command, it will read this variable from its assigned sound and set its "Channel loop enabled" variable to that same value.

Sound loop start	**Initial value:** 0
Format: Integer	**Valid range:** From 0 to 268435455 (*)

It marks the position of the first sample included in the associated sound's looped range. If channel loop is enabled, when playback goes past loop end position, playback will "wrap around" back to this loop start position.

(*) Its maximum allowed value is the last sound position, i.e. Sound Length – 1.

Sound loop end	**Initial value:** (*)
Format: Integer	**Valid range:** From 0 to 268435455 (**)

It marks the position of the last sample included in the associated sound's looped range. If channel loop is enabled, when playback goes past this loop end position, playback will "wrap around" back to the loop start position.

(*) Its initial value is the last sample position in the sound, i.e. Sound Length – 1.
(**) Its maximum allowed value is also the last sample position, i.e. Sound Length – 1.

6.4 Configuration of each sound channel

The variables listed here work the same way as sound variables. The SPU stores a copy of each of these variables for each SPU sound channel. This means there are 16 copies.

Together, each set of these variables describes the current configuration for a single SPU sound channel (see section 3 for an overview of how SPU sound channels are defined). Only one set of these variables is accessible at any time, so each variable in this section is

accessed by control ports via a "pointer" proxy. When the selected sound channel changes, those ports are all redirected to the copy of the variables for the correct channel.

Channel state	**Initial value:** 40h (Channel stopped)
Format: Integer	**Valid range:** Only the values listed

This value is interpreted as the current playback state of its associated sound channel. The possible values are the following:

40h: Channel stopped
41h: Channel paused
42h: Channel playing

Channel assigned sound	**Initial value:** -1
Format: Integer	**Valid range:** -1 to last used sound ID

It represents the sound ID currently assigned to its associated channel. If the channel receives a play command, sound will be generated using the samples from the SPU sound assigned to the channel.

Channel volume	**Initial value:** 1.0
Format: Float	**Valid range:** 0.0 to 8.0

It represents the current output volume level of its associated channel. Its value acts as a linear sample multiplier that will be applied to output samples produced by this channel.

Channel speed	**Initial value:** 1.0
Format: Float	**Valid range:** 0.0 to 128.0

It represents the current playback speed of its associated sound channel. A speed of 1.0 represents unmodified playback, and other values are interpreted as a multiplier. For instance, a speed of 2.0 means the channel position advances 2 sound samples per each output sample. For a value of 0.0, playback will not advance at all.

Channel loop enabled	**Initial value:** False
Format: Boolean	**Valid range:** True / False

It determines if looped playback is currently enabled for its associated sound channel. When true, that channel will keep playing its associated sound's looped region.

Channel position	**Initial value:** 0.0
Format: Double	**Valid range:** 0.0 to (*)

It represents the current playback position of its associated sound channel. This is interpreted as a sample position within its associated sound. When this channel generates its next output sample, it will pick the sample at this position in its associated sound.

Note that the internal format for this variable is a 64-bit IEEE double, which is an exception. This is because it needs to represent positions within a large range while also retaining enough precision for non-integer values. Still, control ports can only handle 32-bit values, so reads and writes from ports will handle channel position as a 32-bit integer.

(*): The upper limit for this variable is different for every channel, and it is updated every time the associated sound changes for that channel. That upper limit will be set to the last sample position in the new associated sound, this is, its sound length – 1.

7 Control ports

This section details the set of control ports exposed by the SPU via its connection to the CPU control bus as a slave device. All exposed ports, along with their basic properties are listed in the following table:

List of exposed control ports			
External address	Internal address	Port name	R/W access
300h	00h	Command	Write Only
301h	01h	Global Volume	Read & Write
302h	02h	Selected Sound	Read & Write
303h	03h	Selected Channel	Read & Write
304h	04h	Sound Length	Read Only
305h	05h	Sound Play With Loop	Read & Write
306h	06h	Sound Loop Start	Read & Write
307h	07h	Sound Loop End	Read & Write
308h	08h	Channel State	Read Only
309h	09h	Channel Assigned Sound	Read & Write
30Ah	0Ah	Channel Volume	Read & Write
30Bh	0Bh	Channel Speed	Read & Write
30Ch	0Ch	Channel Loop Enabled	Read & Write
30Dh	0Eh	Channel Position	Read & Write

7.1 Behavior on port read/write requests

The SPU control ports are not simply hardware registers. The effects triggered by a read/write request to a specific port will often be different than reading or writing values. This section details how each of the SPU ports will behave.

Note that, in addition to the actions performed, a success/failure response will need to be provided to the request as part of the control bus communication. This response will always be assumed to be success, unless otherwise specified. When the provided response is failure, the SPU will perform no further actions and the CPU will trigger a HW error.

Command port

On read requests:
This port is write-only, so a failure response will be provided to the control bus.

On write requests:
The SPU will perform the command execution process detailed in section 8. In all cases the request will be responded with success, even if the command is not recognized.

Global Volume port

On read requests:
The SPU will provide the current value of the internal variable “Global volume”.

On write requests:
The SPU will check if the received value is out of range for the internal variable “Global volume” and, if it is, it will be clamped to range limits. The resulting value will then overwrite the internal variable “Global volume”. This will immediately cause the next sound generations to apply the new global volume.

Selected Sound port

On read requests:
The SPU will provide the current value of the internal variable “Selected sound”.

On write requests:
The SPU will check if the received value corresponds to a currently valid sound ID. In case it is not, the request will be ignored. For valid values, the SPU will overwrite the internal

variable “Selected sound” with the received value. After that, it will redirect all sound configuration ports to point to the set of variables for the new selected sound.

Selected Channel port

On read requests:

The SPU will provide the current value of the internal variable “Selected channel”.

On write requests:

The SPU will check if the received value corresponds to a valid channel ID. In case it is not, the request will be ignored. For valid values, the SPU will overwrite the internal variable “Selected channel” with the received value. After that, it will redirect all channel configuration ports to point to the set of variables for the new selected channel.

Sound Length port

On read requests:

The SPU will provide the current value of the internal variable “Sound length” associated to the currently selected sound ID.

On write requests:

This port is read-only, so a failure response will be provided to the control bus.

Sound Play With Loop port

On read requests:

The SPU will provide the current value of the internal variable “Sound play with loop” associated to the currently selected sound ID.

On write requests:

The SPU will use the received value to overwrite the internal variable “Sound play with loop” associated to the currently selected sound ID. This will immediately cause next sound generations to apply the new loop configuration for the selected sound.

Sound Loop Start port

On read requests:

The SPU will provide the current value of the internal variable “Sound loop start” associated to the currently selected sound ID.

On write requests:

The SPU will check if the received value is out of range for the internal variable “Sound loop start” for this sound and, if it is, it will be clamped to range limits. The resulting value will then overwrite the internal variable “Sound loop start” associated to the currently selected sound ID. This will immediately cause next sound generations to apply the new loop start for the selected sound.

Sound Loop End port

On read requests:

The SPU will provide the current value of the internal variable “Sound loop end” associated to the currently selected sound ID.

On write requests:

The SPU will check if the received value is out of range for the internal variable “Sound loop end” for this sound and, if it is, it will be clamped to range limits. The resulting value will then overwrite the internal variable “Sound loop end” associated to the currently selected sound ID. This will immediately cause next sound generations to apply the new loop end for the selected sound.

Channel State port

On read requests:

The SPU will provide the current value of the internal variable “Channel state” associated to the currently selected channel ID.

On write requests:

This port is read-only, so a failure response will be provided to the control bus.

Channel Assigned Sound port

On read requests:

The SPU will provide the current value of the internal variable “Channel assigned sound” associated to the currently selected channel ID.

On write requests:

The SPU will first check the state of the selected channel ID, and if it is not Stopped the request will be ignored. If the received value is not a valid sound ID the request will be ignored as well. In both cases no further processing will be performed.

If the request is processed, the SPU will use the received value to overwrite the internal variable “Channel assigned sound”. This will immediately cause next drawing operations to apply the new assigned sound for the selected channel.

After that, the SPU will do this processing for the channel’s “Channel position” variable:

- It will set its value to 0
- It will set its upper limit to the new sound’s number of samples – 1.

Channel Volume port

On read requests:

The SPU will provide the current value of the internal variable “Channel volume” associated to the currently selected channel ID.

On write requests:

The SPU will check if the received value is out of range for the internal variable “Channel volume” and, if it is, it will be clamped to range limits. The resulting value will then overwrite the internal variable “Channel volume” associated to the currently selected channel ID. This will immediately cause next sound generations to apply the new volume for the selected channel.

Channel Speed port

On read requests:

The SPU will provide the current value of the internal variable “Channel speed” associated to the currently selected channel ID.

On write requests:

The SPU will check if the received value is out of range for the internal variable “Channel speed” and, if it is, it will be clamped to range limits. The resulting value will then overwrite the internal variable “Channel speed” associated to the currently selected channel ID. This will immediately cause next sound generations to apply the new speed for the selected channel.

Channel Loop Enabled port

On read requests:

The SPU will provide the current value of the internal variable “Channel loop enabled” associated to the currently selected channel ID.

On write requests:

The SPU will use the received value to overwrite the internal variable "Channel loop enabled" associated to the currently selected channel ID. This will immediately cause next sound generations to apply the new loop configuration for the selected channel.

Channel Position port

On read requests:

The SPU will provide the current value of the internal variable "Channel position" associated to the currently selected channel ID. Since this variable is a double, the provided value will be the result of truncating that value to a 32-bit integer.

On write requests:

The SPU will check the number of samples for the SPU sound currently associated to the selected channel ID. The allowed range for this write operation will be determined as: [0 to number of samples – 1], both included. If the received 32-bit integer value is out of range, it will be clamped to range limits. The resulting value will then be converted to a double and used to overwrite the internal variable "Channel position" associated to the currently selected channel ID. This will immediately cause next sound generations to apply the new playback position for the selected channel.

8 Execution of commands

SPU commands are channel playback control instructions that can be requested by the CPU, by sending a value to the SPU Command port. This section describes the SPU behavior when receiving such a request.

There are 6 different commands the SPU can perform. The first 3 operate on a single sound channel, while the other 3 will affect all channels. This table shows their numerical values, as well as the graphic effects that are used for each command.

Command name	Numeric value
Play Selected Channel	30h
Pause Selected Channel	31h
Stop Selected Channel	32h
Pause All Channels	33h
Resume All Channels	34h
Stop All Channels	35h

8.1 Common processing

The general operation described here is common for the execution of all commands.

As a first step, the SPU will check if the requested write value corresponds to one of the valid commands listed above. If it does not, the request will be ignored and no more processing will be done.

8.2 Play Selected Channel command

The SPU will set the currently selected sound channel to state Playing. If the channel's previous state was Stopped, the following actions will be taken to begin a new playback:

1. Channel's position will be set to 0.
2. Channel's "Loop enabled" variable will be set to the same value as the associated sound's "Play sound with loop" variable. This will automatically apply the sound's loop configuration to the channel used to play it.

If the channel's previous state was already Playing, the same 2 actions described will be applied, to cause a retrigger of the sound playback. For a channel previously in Paused state position will not be altered, to resume sound playback.

8.3 Pause Selected Channel command

If the currently selected sound channel is in state Playing, the SPU will set its state to Paused. No actions will be performed for other previous states.

8.4 Stop Selected Channel command

The SPU will set the currently selected sound channel to state Stopped. Stopping an already stopped channel has no effect.

8.5 Pause All Channels command

The SPU will apply the same processing described for the "Pause Selected Channel" command, but the actions will instead be applied to all channels in state Playing.

8.6 Resume All Channels command

The SPU will apply the same processing described for the "Play Selected Channel" command, but the actions will instead be applied to all channels in state Paused.

8.7 Stop All Channels command

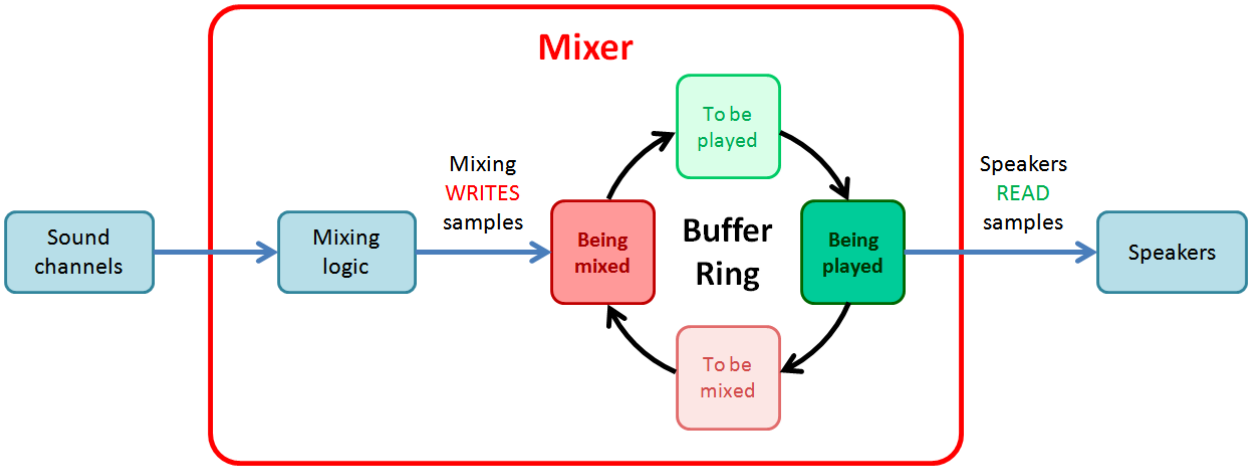
The SPU will apply the same processing described for the "Stop Selected Channel" command, but the actions will instead be applied to all channels.

9 Generation of audio output

As seen in section 5, the Vircon32's SPU generates audio on a frame per frame basis, governed by console's global timing at 60 Hz. However, sound playback devices such as sound cards and speakers will usually require that each single sample is provided to the speakers individually at the correct moment, at the playback rate of 44100 Hz.

To account for the timing difference between sound generation and output, the samples that are generated at each frame will need to be stored in one or more "waiting buffers" until the time is right for each individual SPU sample to be sent for playback. There can be many possible strategies for this, depending on the audio system implementation and its output signal format.

One of the possible strategies, common in software audio handling, is called the "buffer ring". It consists in having several output buffers, that are cyclically played and refilled with new samples. Note that for this to work correctly each buffer will need store samples for 1 frame. So, the size of the buffer will be 44100 / 60 = 735 frames. The image below shows an example of how a 4-buffer ring could be implemented:



9.1 Sound buffers and latency

In some real-world implementations, the console's global 60 Hz timing could be imprecise. So, strategies like the buffer ring will need to use buffers in a way that ensures that:

1. The speakers never run out of sound samples.
2. The mixer always has a free buffer to generate new sound.

In general, audio buffer strategies will always need to face a trade-off. On one hand, using more buffers allows these 2 conditions to still be met even with more imprecise timings. But the more buffers are used, the greater sound latency there will be, since using more buffers makes playback queues grow in size.

9.2 Output audio signal

Depending on the implementation's audio signal format, the output signal will need to either separate the stereo output into its 2 channels (i.e. break the 32-bit SPU samples into 2 separate 16-bit sample sequences for left and right), or arrange them in a joint sequence with well identified individual samples for each speaker.

The SPU will then transfer that information to the speakers via the audio output signal. If the signal format/protocol needs it, timing or sequencing information will be added to form a valid audio sample transmission.

Implementations are free to choose the communication format and physical connectors for audio and video signals. They may be sent separately or, as is common in most displays nowadays, use a joint connector that sends both of them to the same display.

10 Responses to control signals

As with all components in the console, whenever a control signal is triggered the SPU will receive it and produce a response to process that event. For each of the control signals, the SPU will respond by performing the following actions:

Reset signal:

- If any audio playback was still in progress, it is stopped.
- All SPU internal variables are set to their initial values. This includes configuration variables for all sounds and all channels.
- Any additional effects associated to those changes in internal variables are immediately applied, as described in the control port write behaviors.
- All samples in output buffers are set to 0.

Frame signal:

- The SPU generates output for 1 new frame, as described in section 5.

Cycle signal:

- The SPU does not need to react to this signal, unless specific implementation details require it.

In addition to reacting to control signals, the SPU will also need to perform the following processing in response to console-level events:

When a new cartridge is connected:

- The SPU will locate the sounds contained in the BIOS and, if present, the cartridge.
- The SPU will perform the connection process to described in section 3 to assign a sound slot to each sound, and gain access to their sample information.
- The upper limit for variable “Selected sound” will be adjusted to the last used sound slot ID.
- For all used sound slots, the value for their variable “Sound length” will be set to the number of samples of their assigned sound.
- For all used sound slots, the upper limit for their variables “Sound loop start” and “Sound loop end” will be set to the number of samples of their assigned sound – 1.

(End of part 5)