Vircon32

32-BIT VIRTUAL CONSOLE

System specification

Part 8: External elements

Document date 2022.12.31 Written by Carra

What is this?

This document is part number 8 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 orded 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 emulators.

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.

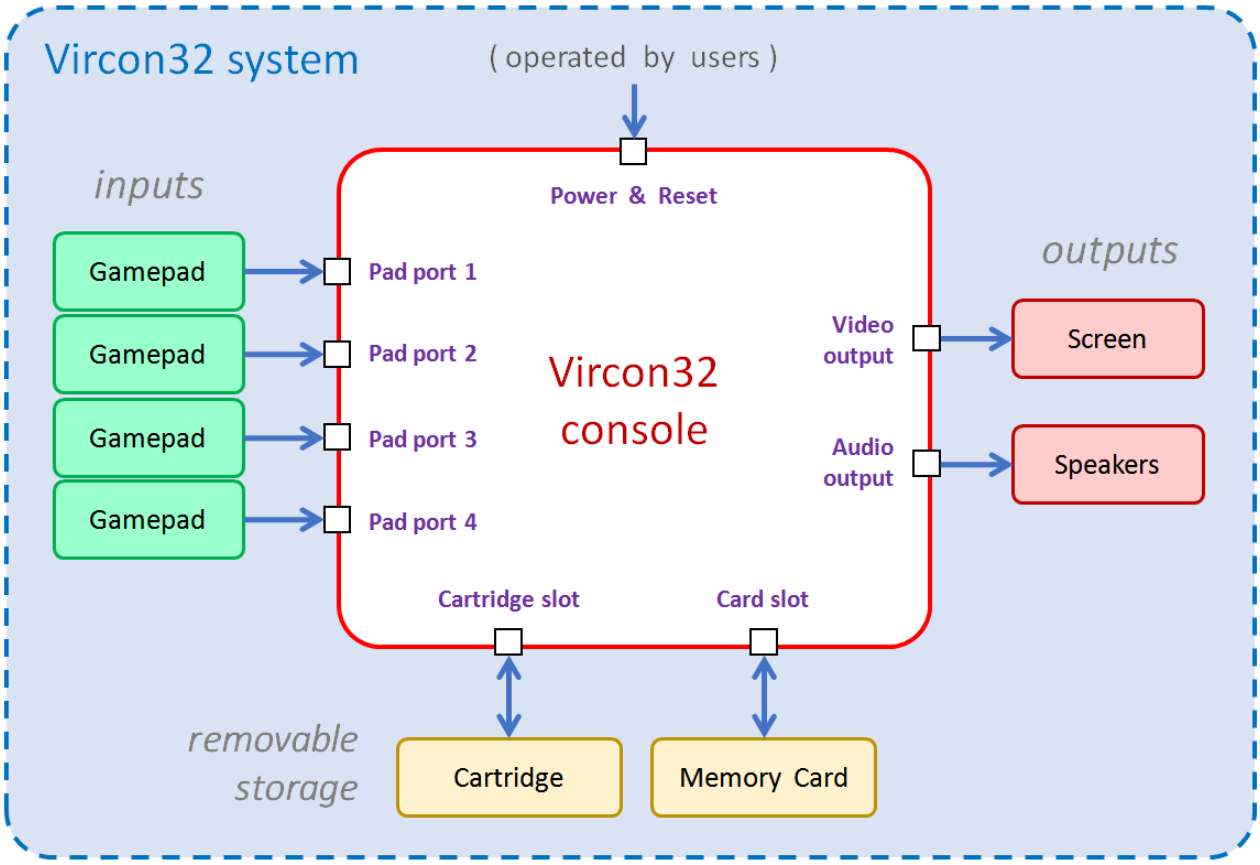
About this document

Summary

Previous specification documents already defined all elements within the console. We now need to describe components that, being still part of the Vircon32 system, are external to the console itself. Part 1 of the specification already presented a basic overview of these external elements, and now part 8 will provide a complete description.

1 The complete system

A Vircon32 system does not only include the console itself: other components external to the console are also required for it to function. A complete Vircon32 system will then consist of the console along with all possible external elements connected to it. We can see those elements and their connections in this diagram:



The console is the central element of this system, and every external element is conceived to interact exclusively with it. For all external elements, next sections will detail their functions and behavior, as well as their communication with the console.

1.1 Removable external elements

In general, all external devices are considered to be removable: it is conceivable (although not very useful) for the console to be turned on with none of them present. There could also be any combination of present and absent devices.

This means that, to be fully compliant, implementations must support the option of attaching and detaching every external device. For software implementations this should be trivial in most cases. However, for some types of hardware implementations this might not be possible.

While so far Vircon32 hardware has been depicted as a home console, there could be hardware versions in other formats, such as arcade cabinets or handheld devices. In these it can be acceptable for an implementation that the screen, speakers and/or gamepads are fixed and cannot be detached or disconnected. An example of this could be this concept of a Vircon32 arcade cabinet:

In a system like this the screen, the speakers and the 4 gamepads would be embedded into the cabinet. That means they will always be present and remain connected. This is acceptable, as long as the system supports all the elements of a complete system. If instead this arcade cabinet was thinner and only included 2 joysticks, it would not be considered fully compliant unless 2 additional gamepads could be connected in some way.

As for the removable storage components (cartridges and memory cards), they are required to be removable in any implementation because they are meant to be interchangeable. This can also be seen in the images, where both of these elements are inserted via slots.

2 Gamepads

Gamepads allow players to interact with the game by pressing directions and buttons. The console will read their state every frame, and games will use this information to perform actions or make decisions. Up to 4 gamepads can be connected to the console.

Vircon32 gamepads are all considered identical in capabilities and layout. The console cannot distinguish between them, other than the specific port each one is connected to.

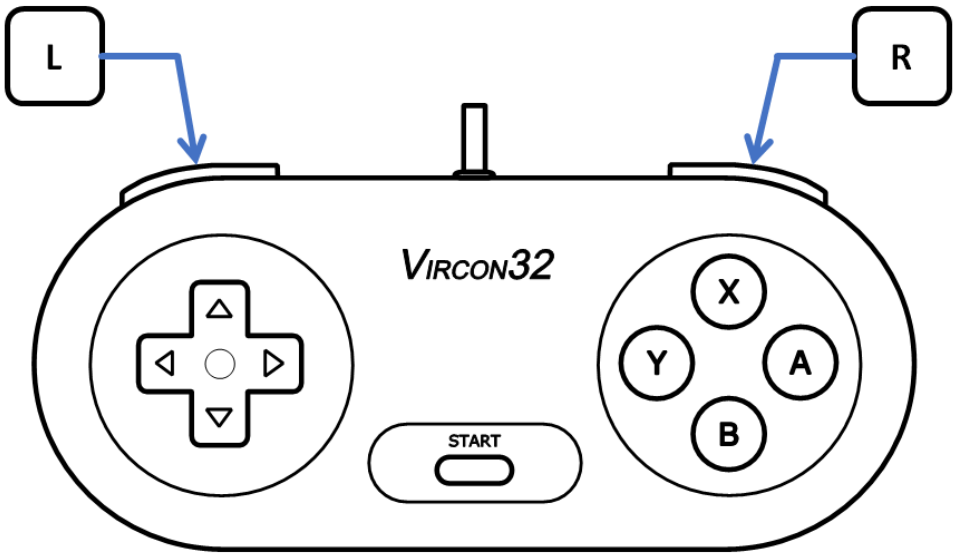
2.1 Controls and layout

Each gamepad features 11 digital controls. These are split into 2 categories (4 directions and 7 buttons) as shown in these lists:

Directions:	**Buttons:**
Left, Right, Up, Down	A, B, X, Y, L, R, Start

Each control only has 2 possible states, and gamepads represent them with 1 bit each, being 1 = pressed and 0 = unpressed. For all controls their neutral state is unpressed: each control only becomes pressed when users take some active action, such as applying a force.

The Vircon32 gamepad physically organizes its controls in separate groups: a directional pad (Left, Right, Up, Down), 4 front buttons (A, B, X, Y), 2 shoulder buttons (L, R) and a central button (Start). For all implementations, the controls on each gamepad must follow the layout and placement shown in this image.



This means that game creators can safely assume this control layout when deciding how to implement their control scheme. For instance, a shooting game could assume the diamond shape of the 4 front buttons to make them work as a "secondary directional pad" that controls which direction to shoot at (X shoots up, Y shoots left, etc).

Vircon32 gamepads do not include any additional mechanism or automation for gamepad controls. This means they feature no turbo-fire, auto-fire, auto-hold, or programmability of any kind.

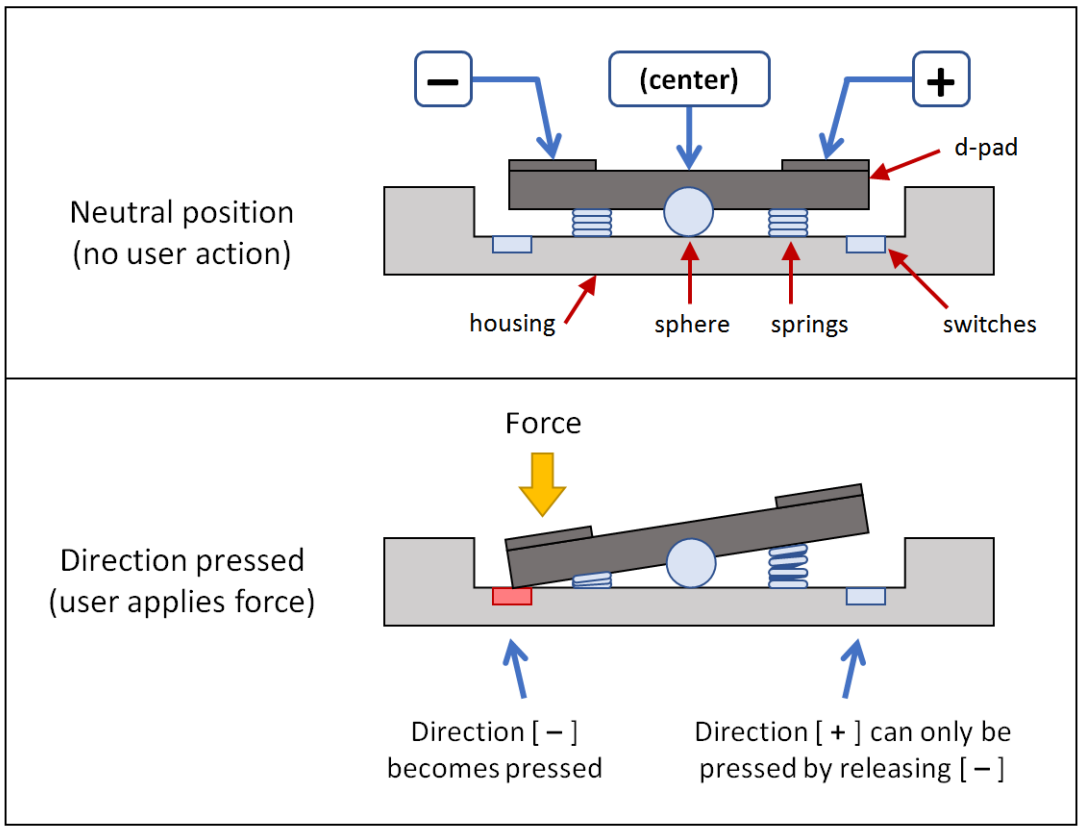
2.2 Directional pad (d-pad)

The directional pad can be implemented in different ways, to be chosen by each implementation. Possiblr options are: 4 separate buttons, a cross shape, an octogon for explicit diagonals, etc. For any chosen option, its output must always be 4 bits (one for each of the 4 directions).

All Vircon32 d-pads are required to implement some mechanism to prevent that opposite directions can be pressed at the same time. This is needed to ensure that the console can always interpret a consistent d-pad direction. This means that:

- The d-pad's horizontal axis must prevent pressing both Left and Right at once.
- The d-pad's vertical axis must prevent pressing both Up and Down at once.

This can be implemented with a variety of solutions. As an example, classic consoles normally used a tilting mechanism. A simple diagram of this is shown in the image below. If we consider both ends of a same axis, called here **[–]** and **[+]**, we would have:



2.3 Connection to the console

Communication from the gamepads to the console is conceived as completely unidirectional: the console cannot influence gamepad behavior or state in any way. The console will only read the state of gamepads through each of the ports.

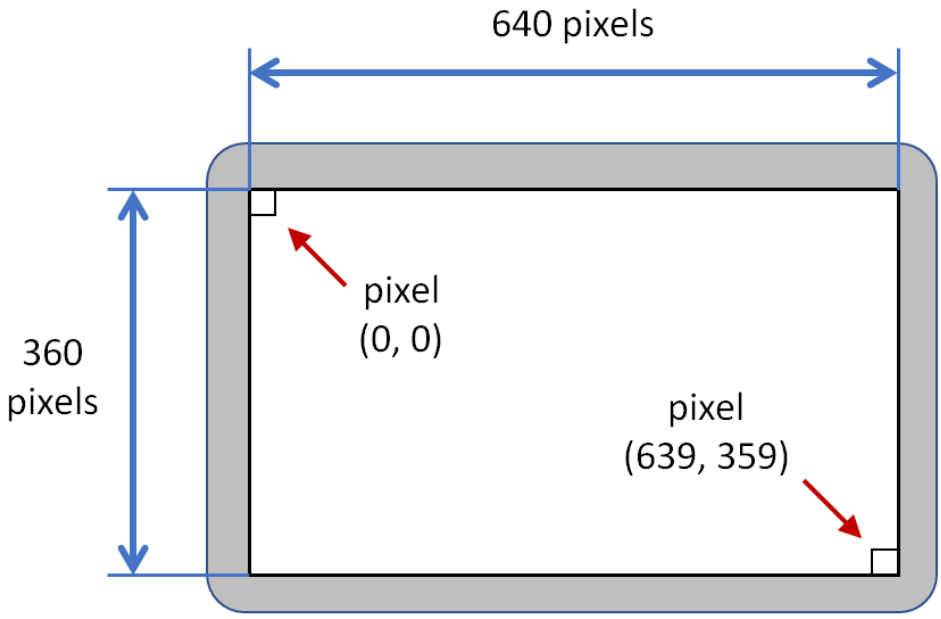
Implementations can choose any wirings and connectors, as well as any protocol for electrical signals and data transmission. However, the chosen connectors must include some kind of signal or sensor that allows the console to know at any moment if a gamepad is connected to each of the 4 ports. This information is represented as a single bit: 1 = connected and 0 = disconnected.

3 Screen

The screen is used to display the image produced by the console and make it visible for the players. It will receive the pixel color information sent to it by the GPU every frame, and use it to produce the required image.

3.1 Screen dimensions

The Vircon32 screen is a rectangular matrix of 640 pixels horizontally x 360 pixels vertically (a total of 230400 pixels). Pixels are square (1:1 aspect ratio) and have no gaps between then, so the global aspect ratio for the full screen results in 16:9. Display size is irrelevant for our purposes, and can be freely chosen by each implementation.



Screen pixels are considered numbered from top-left (0,0) to bottom-right (639,359). The screen does not feature different possible video modes or display configurations: image properties will be the same for all games across all implementations.

The Vircon32 screen is always assumed to be viewed in a fixed, horizontal position. Game creators cannot assume that the screen can be rotated in any way, since implementations are not required to have this capability. This means, for instance, that no Vircon32 game should require users to rotate the screen 90 degrees, in order to have a vertical image ratio of 9:16.

3.2 Image quality

The screen displays colors in the RGB space, having a range of 24 bits (true color). For each pixel there are 3 channels for red, green and blue. Each channel has a color depth of 8 bits, going from 0 (pure black) to 255 (full intensity). The screen produces video only with the received color for each pixel: the final image has no color blending or filtering.

The screen displays 60 frames per second. It always produces video using full frames: there is no interpolation or interlacing effects. Real screens will always have a certain input lag, so implementations have to decide how to manage this.

3.3 Connection to the console

Communication from the console to the screen is conceived as completely unidirectional: the screen cannot influence the console's behavior or state in any way. The console will only send the video signal through its connector, and has no way to even determine if there is a screen connected to it.

Implementations can choose any wirings and connectors, as well as any protocol for electrical signals and data transmission. However, the chosen communication needs to include a way to ensure the correct timing for each frame.

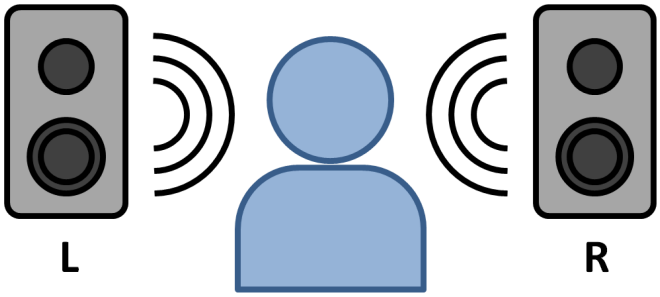
As stated in the GPU and SPU documents, it is common in practice for a display to also integrate audio capabilities. Current communication protocols such as HDMI can join audio and video in the same signal and therefore share wirings and connectors. So, even if the screen and the speakers are being treated as separate entities, it would be valid for an implementation to join audio and video outputs into a combined signal sent to a single device.

4 Speakers

The speakers are used to play the sound generated by the console and make it audible for the players. They will receive the continuous sequence of samples sent to them by the SPU, and use it to produce the required sound.

4.1 Sound channels

Vircon32 uses a standard set of 2.0 speakers to produce stereo sound. This means there are 2 independent sound sources with different placement relative to the players: one at their left and another another at their right.



In this document we will treat each sound source as a separate speaker, but there can be implementations where a single device integrates 2 separate sources for left and right.

4.2 Sound quality

Sound is produced at a sampling rate of 44100Hz, and samples have 16-bit precision. Each speaker will receive a constant, separate sequence of samples sent to it by the SPU, and use it to produce the required sound for that channel (Left or Right).

While headphones can be used as speakers, there must always exist a sound source that can be heard by all 4 players. There should be no differences in sound quality, lag, or distortions between the sound heard by any of the players.

Vircon32 speakers produce sound only with the received values for each sample in the sequence: the final sound has no filters or equalization adjustments.

4.3 Connection to the console

Communication from the console to the speakers is conceived as completely unidirectional: the speakers cannot influence the console's behavior or state in any way. The console will only send the audio signal through its connector, and has no way to even determine if there is a set of speakers connected to it.

Implementations can choose any wirings and connectors, as well as any protocol for electrical signals and data transmission. However, the chosen communication needs to include ways to separate the sequences for Left and Right speakers, and to ensure the correct timing for each sample.

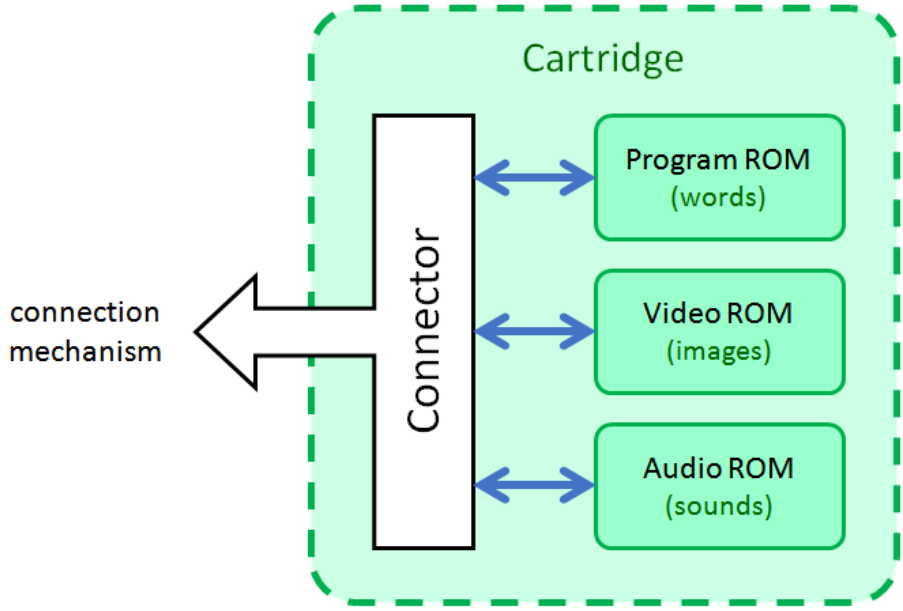
As already mentioned for the screen, even if the screen and the speakers are being treated as separate entities, it would be valid for an implementation to join audio and video outputs into a combined signal sent to a single device.

5 Cartridges

Cartridges are the main storage support for Vircon32. In general each cartridge will store a separate game and the console will run its program from the cartridge. Cartridges can each have different contents but they all have the same structure and connectors to make them interchangeable.

5.1 Structure of a cartridge

Vircon32 cartridges can contain up to 3 independent read-only memories (ROMs): program, video and audio. Communication from the console will take place through some kind of connector that will carry information about all cartridge ROMs and manage access to their contents.



All cartridges are required to contain a program ROM, while audio and video ROMs are both optional. Each of the 3 ROMs, when present, can have a variable number of elements (words / images / sounds) in different cartridges.

The console must have some way (chosen by the implementation) to determine the actual contents of each cartridge. This can be done either by including additional information to cartridges (headers or metadata), or by designing console-cartridge communication in a way that allows the console to determine the inner structure for every particular cartridge.

All 3 ROMs store their contents without using any compression, and for each of them the capacity limit is the following:

- Program ROM can store up to 128 MWords = 128 x 1024 x 1024 words = 512 MB.
- Video ROM can store up to 256 textures = 256 x 1024 x 1024 pixels = 1 GB.
- Audio ROM can store up to 1024 sounds, and 256 x 1024 x 1024 samples = 1 GB.

This means that the maximum possible storage for a cartridge is a total of 2.5 GB.

5.2 Memory performance

While no specific performance levels will be stated, the 3 cartridge ROMs are assumed to be fast enough to be capable of the following:

- Every cycle, program ROM memory can provide the CPU with all words it requests in time for it to finish the current instruction within that same cycle.

- Every frame, video ROM memory can provide the GPU with all pixels it needs in time for it to finish all drawing operations within that same frame.

- Every frame, audio ROM memory can provide the SPU with all samples it needs in time for it to finish all sound generation within that same frame.

5.3 Connection to the console

Communication from cartridges to the console is conceived as completely unidirectional: cartridges are passive components and, since all their memory is read-only, the console can only read their contents but not influence their state in any way.

Implementations can choose any wirings and connectors, as well as any protocol for electrical signals and data transmission. However, there are some requirements that the chosen connectors and/or protocols must meet:

- They must include some kind of signal or sensor that allows the console to know at any moment if a cartridge is connected to its cartridge slot. This information is represented as a single bit: 1 = connected and 0 = disconnected.

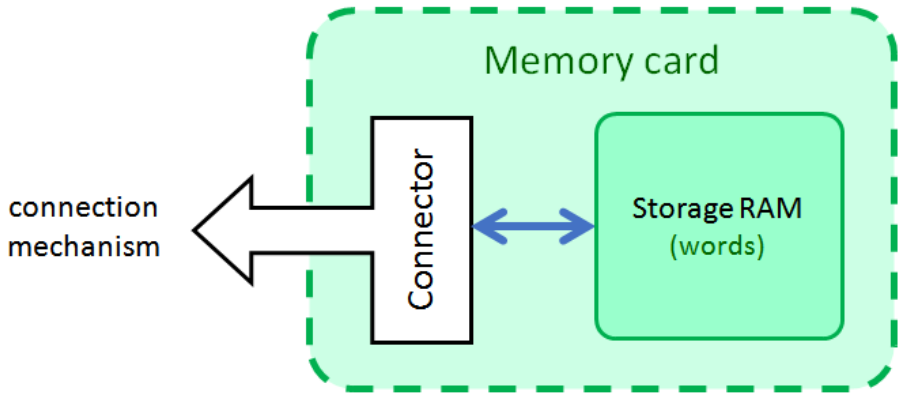
- The chosen communication needs to be the same for all cartridges designed for that particular implementation, in order to ensure they are all interchangeable.

6 Memory cards

Memory cards provide the console with a small permanent storage, which games can use to use to load and save data between different sessions. Unlike cartridges, all Vircon32 memory cards are all considered identical. The only way for the console to distinguish between memory cards is by examining their contents.

6.1 Structure of a memory card

All Vircon32 memory cards are functionally identical, and as such they all have the same structure and storage capacity. A memory card has an internal random-access memory (RAM) that allows any of its contained words to be read and written independently. Communication with the console will take place through some kind of connector that will manage access to the internal memory.



Unlike console RAM, memory card RAM stores its contents permanently. They will not be deleted under a power off/reset, or when the card is extracted from the console. Vircon32 memory cards store their contents without using any compression, and they all have a fixed capacity of 256 KWords = 256 x 1024 words = 1MB.

6.2 Memory performance

While no specific performance levels will be stated, card memory is assumed to be fast enough so that every cycle, it can respond to all of the CPU's requests in time for it to finish the current instruction within that same cycle.

6.3 Connection to the console

Communication from memory cards to the console is bidirectional. Although a memory card is a passive component (so communication is always triggered from the console), card memory can be read and written. So, unlike cartridges, it is possible for the console to modify the contents of a memory card.

Implementations can choose any wirings and connectors, as well as any protocol for electrical signals and data transmission. However, the chosen connectors must include some kind of signal or sensor that allows the console to know at any moment if a memory card is connected to its card slot. This information is represented as a single bit: 1 = connected and 0 = disconnected. Also, the chosen communication needs to be the same for all memory cards designed for that particular implementation, in order to ensure they are all interchangeable.

(End of part 8)