Vircon32

32-BIT VIRTUAL CONSOLE

System specification

Part 6: Controller chips

Document date 2022.12.17 Written by Carra

What is this?

This document is part number 6 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 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

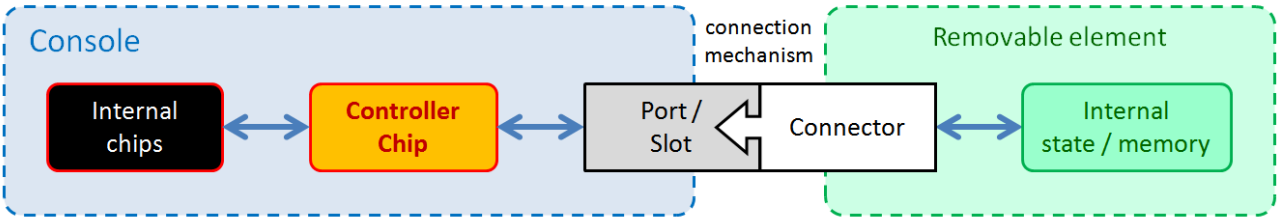
Summary

Part 6 of the specification defines the console's 3 controller chips. For each of them this document will describe its behavior, its control ports, its internal variables and the general process of its operation.

1 About controller chips

A Vircon32 console will need to interact with some external elements (gamepads, cartridges and memory cards) that are removable, and can be present or absent at any given moment. Because of this, there cannot be direct connections from internal console components to these removable elements. Instead, it is needed to have "controller" chips that will act as a proxy in their communications.

Removable elements get connected to the console using some kind of logical or physical mechanism that allows the console to determine when they are present.



The base function of controller chips is provide a safe way for the CPU to communicate with those possible external elements, by allowing it to:

1. Determine when a given external element is present.
2. Receive an error response when attempting to access an absent element.
3. Access their memory or state when the element is present.

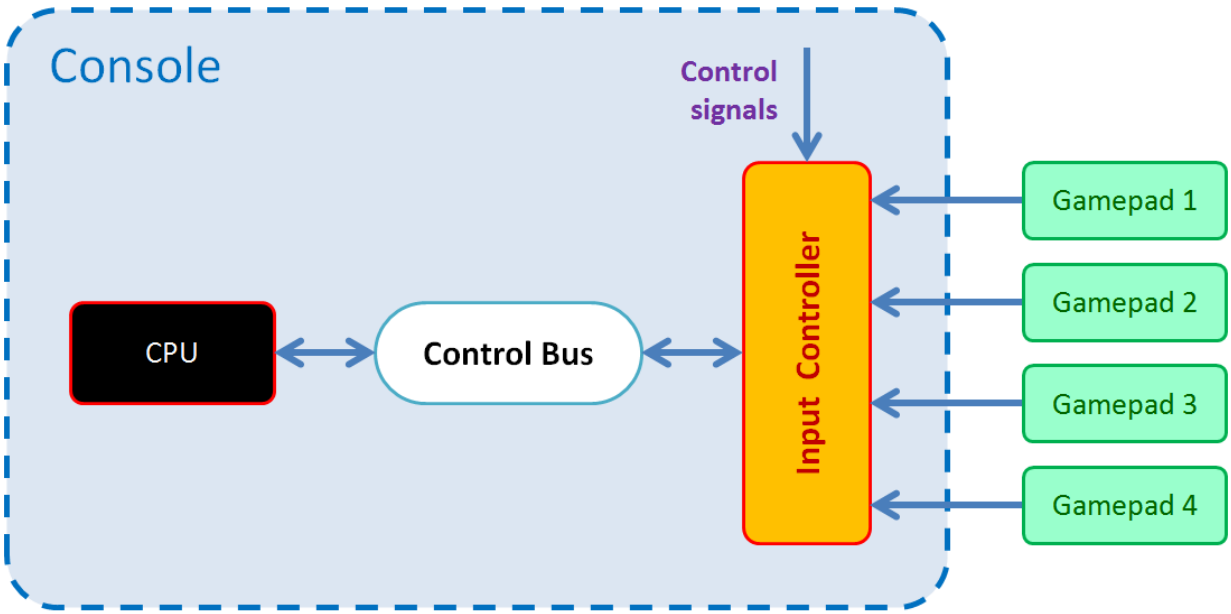
There are 3 controller chips: Input controller, Cartridge controller and Memory card controller. The following sections will each describe one of these chips in detail. To group the 3 chips into this single document, each section for a particular chip will instead be done as a subsection in this document unlike the previous ones.

2 Input controller

The input controller is the chip in charge of controlling the console's 4 gamepad ports. All Vircon32 gamepads are identical but the gamepad ports are numbered. For instance: even if only 1 gamepad is connected, it will be treated as gamepad 2 if connected to the second port. This results in a total of 16 combinations (2^4).

2.1 External connections

Since the input controller is just one of the chips forming the console, it cannot operate in isolation. This figure shows all of communications its with other components. As mentioned, the 4 connections to gamepads are numbered and non interchangeable.



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

2.1.1 Control signals

As all console components, the input controller receives the signals for reset, new frame and new cycle. The responses to those signals are detailed in section 2.6 of this document.

2.1.2 Control Bus

The input controller is connected as slave device to the Control Bus, with device ID = 4. This allows the bus master (the CPU) to request read or write operations on the control ports exposed by the input controller. The list of its ports as well as their properties will be detailed in later sections.

2.1.3 Gamepads

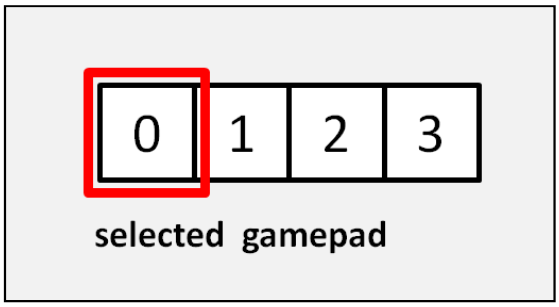
The input controller manages 4 numbered gamepad ports. Each port can be queried by the CPU to determine if it currently has a gamepad connected to it. When a port has a connected gamepad, the input controller will be able to read its current state.

2.2 Working concepts

Before explaining the input controller's functions or the internal variables that affect them, we must present the basic concepts that this chip is built around.

2.2.1 Selected gamepad

All available gamepad ports form a range of usable gamepad IDs from 0 to 3. To determine which gamepad to use when providing control states or applying internal variables, the input controller always considers one of those IDs as "selected". The gamepad selected by default is the first, corresponding to gamepad ID = 0.



2.2.2 Gamepad controls

Vircon32 gamepads are all identical. Each gamepad features 11 controls: 4 directions and 7 buttons. These lists enumerate all of them:

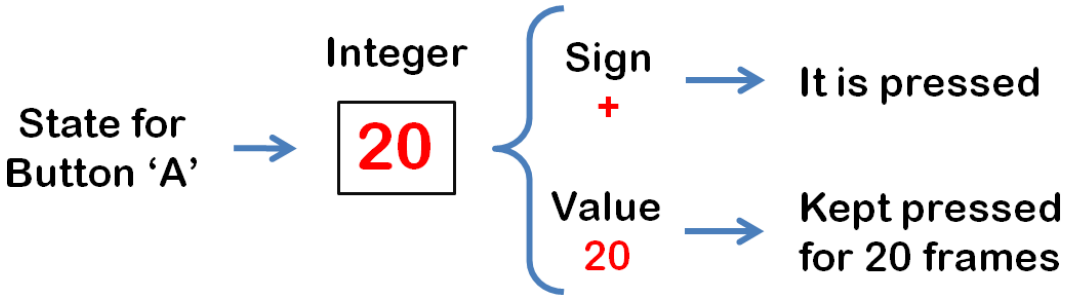
Directions:	**Buttons:**
Left, Right, Up, Down	A, B, X, Y, L, R, Start

These controls are all digital and gamepads represent their state as a single bit: 1 (True) when pressed, 0 (False) when not pressed.

2.2.3 Gamepad state format

Even though every gamepad control can be represented in 1 bit, the input controller actually stores those states using integer format. The reason for this is that the input controller does not just store control state as it is read from gamepads, but instead these values are processed over time to provide more information than just the current state of each control.

The input controller keeps track of past states to also report the time (in frames) that the control has been in the present state. This is encoded as the integer's sign and value, being positive sign for pressed and negative for unpressed. For instance, if button A has a current state of 20, its interpretation will be the following:



Games will often need to perform an action only once at the instant a button becomes pressed. By providing this extra information, the input controller makes possible to instantly know if a pressed button has just been pressed, instead of having to store previous states to determine this.

Control state values are guaranteed to never be zero so that a sign can always be determined. To ensure that, these 2 rules are implemented:

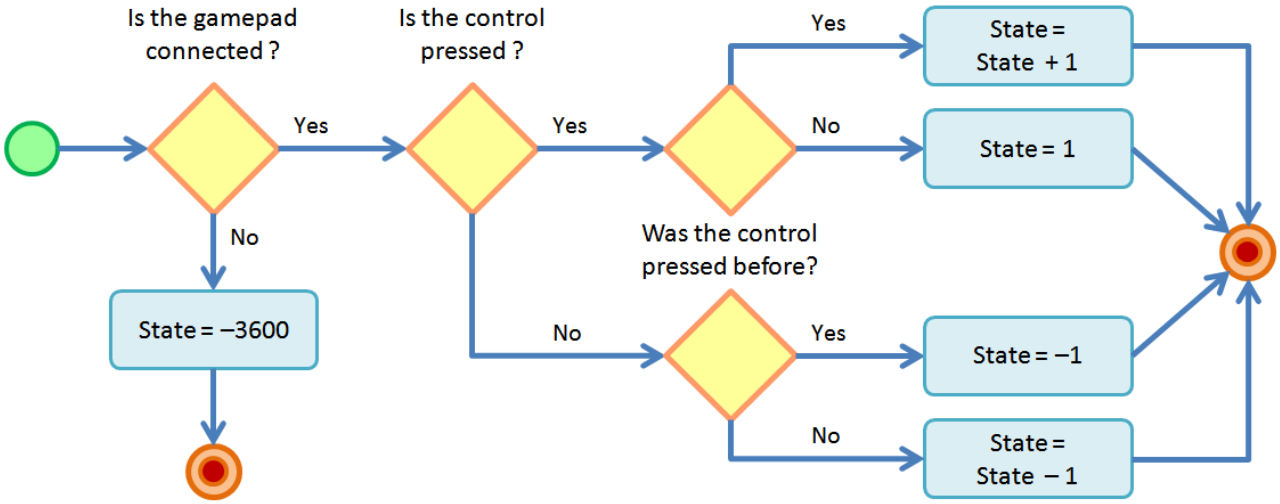
1. Controls that just changed state in the present frame will be set to value +/- 1 and begin their frame count from there if their state is maintained.

2. If a given gamepad port has no gamepad connected to it, the input controller will set all its control states to their lower bound of -3600. This value could be interpreted as the control not being pressed for a long time.

2.3 Gamepad reading process

The gamepad reading process is the process used by the input controller to read the current state of gamepads and update its internal variables in consequence. The CPU can then request to read those internal variables to determine the current gamepad inputs.

As most console processes, timing for reading gamepads is based on frames. The gamepad reading process is triggered at the beginning of each frame and, for every execution, it will perform a loop over each of the 4 gamepad ports. Then, for each gamepad port, it will perform the following processing to for each of the 11 controls in each gamepad to determine their respective updated states (stored in their respective internal variables).



Note that, due to range restrictions, control states cannot be incremented beyond 3600, or decremented beyond -3600 so they will be clamped accordingly. In addition to updating control states with this algorithm, the input controller will also update the boolean variables representing whether each gamepad is currently connected.

2.3.1 Direction coherence

Vircon32 requires its gamepads to physically prevent opposing directions in a same axis from being pressed at once. This means that pressing up + down, or left + right should not be possible. Still, in some implementations it may not be possible to guarantee this for every possible gamepad.

In case this gamepad feature cannot be ensured, the input controller will be required to filter direction pressing events or states to ensure that programs will never encounter an incoherent direction in the directional pad. A common way to do this is to consider that when a direction control becomes pressed, the opposite direction becomes automatically released. In other words, the last pressed direction in each axis is given priority.

2.3.2 Gamepad connection events

When a gamepad is connected or removed, the input controller does not provide any signal to report this event on the fly. The console will only know about this when, after gamepad states are read for the next frame, there is a change in the connected variable for a given gamepad port. Implementations can internally work using these events if needed.

2.4 Internal variables

The input controller 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, boolean, etc) described in part 2 of the specification. Here we will list and detail all internal variables, organized into sections.

2.4.1 Global variables

Selected gamepad	**Initial value:** 0
Format: Integer	**Valid range:** From 0 to 3

This value is the numerical ID of the currently selected gamepad. The selected gamepad is the one that will be used in all gamepad related operations.

Note that the 4 gamepad IDs are always selectable, even if their corresponding gamepads are currently not connected.

2.4.2 State of each gamepad

The variables listed here are special. The input controller stores a copy of each of these variables for each existing gamepad port. This means there are 4 copies. All 4 of them always exist, independently of the currently connected gamepads.

Together, each set of these variables describes the current inputs for a single gamepad. 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 gamepad changes, those ports are all redirected to the copy of the variables for the correct gamepad.

Gamepad connected	**Initial value:** (*)
Format: Boolean	**Valid range:** True/False

This value indicates whether the associated gamepad is connected or not.

(*) Its initial value is determined by the presence or absence of a gamepad in the associated gamepad port on console startup.

Gamepad {*control*}	**Initial value:** -3600
Format: Integer	**Valid range:** -3600 to 3600 (except 0)

These are 11 variables, corresponding to each of the 11 gamepad controls listed in section 2.2.2. Each of them represents the current state for that control in the associated gamepad. This state is the already processed state, not just the boolean state. It is therefore interpreted as {sign, value} as described in section 2.2.3.

2.5 Control ports

This section details the set of control ports exposed by the input controller 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
400h	00h	Selected Gamepad	Read & Write
401h	01h	Gamepad Connected	Read Only
402h	02h	Gamepad Left	Read Only
403h	03h	Gamepad Right	Read Only
404h	04h	Gamepad Up	Read Only
405h	05h	Gamepad Down	Read Only
406h	06h	Gamepad Button Start	Read Only
407h	07h	Gamepad Button A	Read Only
408h	08h	Gamepad Button B	Read Only
409h	09h	Gamepad Button X	Read Only
40Ah	0Ah	Gamepad Button Y	Read Only
40Bh	0Bh	Gamepad Button L	Read Only
40Ch	0Ch	Gamepad Button R	Read Only

2.5.1 Behavior on port read/write requests

Control ports in the input controller are not simply hardware registers. The effects triggered by a read/write request to a specific port can be different than reading or writing values. This section details how each of the input controller 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 input controller will perform no further actions and the CPU will trigger a HW error.

Selected Gamepad port

On read requests:
The input controller will provide the current value of the internal variable "Selected gamepad".

On write requests:
The input controller will check if the received value corresponds to a valid gamepad ID. In case it is not, the request will be ignored. For valid values, the input controller will overwrite the internal variable "Selected gamepad" with the received value. After that, it will redirect all gamepad configuration ports to point to the set of variables for the new selected gamepad.

Gamepad Connected port

On read requests:
The input controller will provide the current value of the internal variable "Gamepad connected" 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.

Gamepad { *control* } port

This same behavior applies to the 11 ports (with addreses 02h to 0Ch), that correspond to the 11 gamepad controls listed in section 2.2.2.

On read requests:
The input controller will provide the current value of the corresponding internal variable "Gamepad { control }" associated to the currently selected gamepad ID.

On write requests:
These ports are read-only, so a failure response will be provided to the control bus.

2.6 Responses to control signals

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

Reset signal:

- All input controller internal variables are set to their initial values. This includes configuration variables for all gamepads.
- For each gamepad port, the input controller will check if there is a gamepad connected and update its corresponding “Gamepad connected” variable.
- Any additional effects associated to those changes in internal variables are immediately applied, as described in the control port write behaviors.

Frame signal:

- The input controller triggers the gamepad reading process, as described in section 2.3.

Cycle signal:

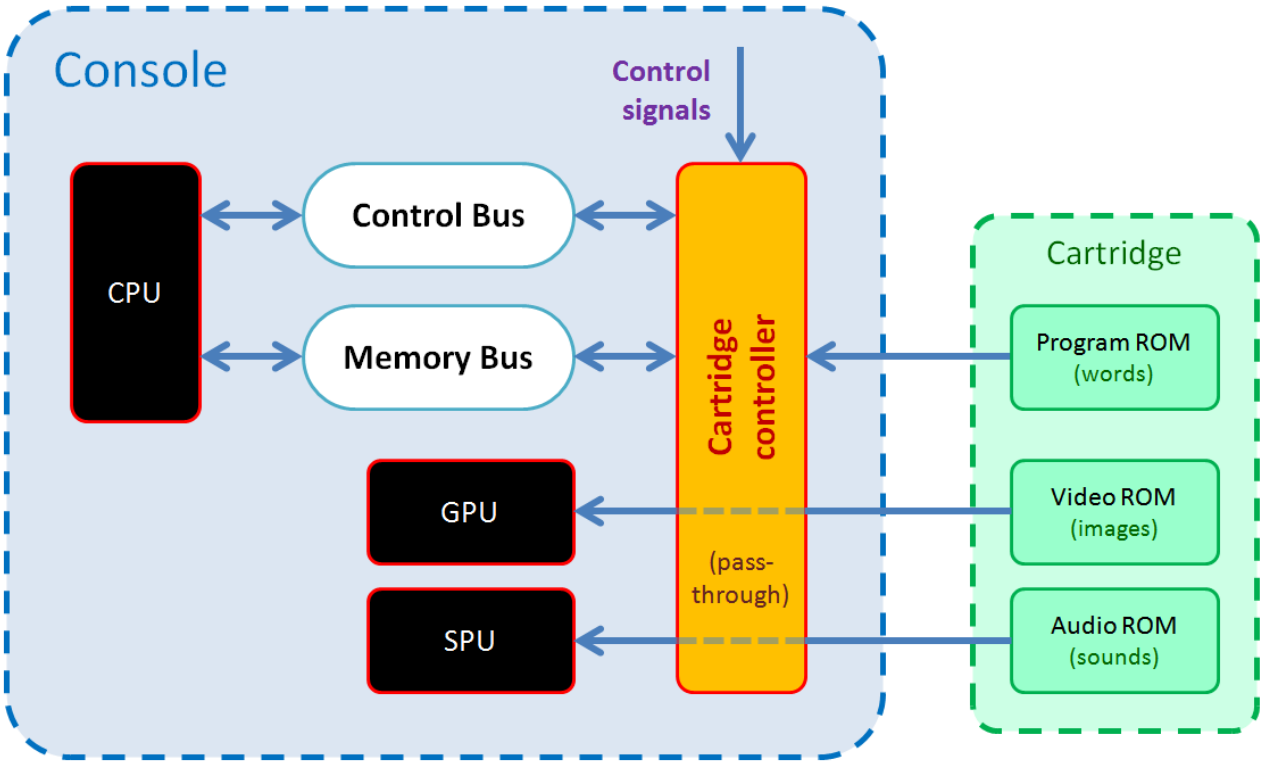
- The input controller does not need to react to this signal, unless specific implementation details require it.

3 Cartridge controller

The cartridge controller is the chip in charge of controlling the console's cartridge slot. Unlike gamepads, only 1 cartridge can be present. However, each cartridge will have a different content: out of the 3 cartridge ROMs (program, video and audio) only the first will always exist, and all of them may contain a different number of elements every time.

3.1 External connections

Since the cartridge controller is just one of the chips forming the console, it cannot operate in isolation. This figure shows all of communications its with other components. As mentioned, the connected cartridge (if any) could be a diferent one every time.



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

3.1.1 Control signals

As all console components, the cartridge controller receives the signals for reset, new frame and new cycle. The responses to those signals are detailed in section 3.5 of this document.

3.1.2 Control Bus

The cartridge controller is connected as slave device to the Control Bus, with device ID = 5. This allows the bus master (the CPU) to request read or write operations on the control

ports exposed by the cartridge controller. The list of cartridge controller ports as well as their properties will be detailed in later sections.

3.1.3 Memory Bus

The cartridge controller is connected as slave device to the Memory Bus, with device ID = 2. This allows the bus master (the CPU) to request read or write operations on the memory addresses exposed by the cartridge. The range and properties of cartridge memory addresses will be discussed in later sections.

3.1.4 Cartridge

The cartridge controller manages the cartridge slot. This slot can be queried by the CPU to determine if it currently has a cartridge connected to it. When a cartridge is present, the cartridge controller will be able to read its contents.

The cartridge slot features a safety mechanism that blocks it as long as the console is powered on. For this reason, it is safe to assume that cartridge connection and contents will not change during console operation.

3.1.5 GPU and SPU

The GPU and SPU need to access the cartridge's video and audio ROM respectively. Since these ROMs may not be present, those 2 connections have to be done via the cartridge controller. This is needed for each chip to determine that its target ROM is present, and the number of elements it contains.

Once those initial informations are known, connections from GPU and SPU are conceived as simple pass-throughs: each chip is able to read those contents freely. Still, the actual mechanisms to set up and manage this access are to be defined by the implementation.

3.2 Cartridge memory

The console does not itself contain any program for the CPU other than the BIOS routines, so it needs to read the memory words stored in cartridges by connecting to their program ROM. A program ROM is a read-only memory region that contains a sequence of 32-bit words.

The cartridge controller is connected to the memory bus to allow any connected cartridge to expose the contents of its program ROM (which always exists) to the CPU. The cartridge controller uses device ID = 2 within the memory bus so, for a cartridge program ROM containing N words, the range of addresses for cartridge memory will be:

Internal addresses → From 0 to N – 1
External addresses → From 20000000h to (20000000h + N – 1).

The size of cartridge program ROM will vary for each cartridge. It can contain any number of words from 1 up to the cartridge program ROM size limit of 128 x 1024 x 1024.

3.2.1 Connection to program ROM

When a cartridge containing N words is connected, the first N internal memory addresses for the cartridge controller from 0 become assigned to each of the words, in order. The rest of addresses up to (128 x 1024 x 1024 – 1) will become unused and not accessible.

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 cartridge controller to establish this connection until the next console power on.

It is up to the implementation to decide how to establish a connection to the program ROMs and locate and read their words. For example, it is possible to read the whole program ROM beforehand upon connection. Another option would be to keep a pointer to cartridge memory and have the cartridge controller will read words on the fly.

3.2.2 Behavior on memory read/write requests

Any memory address within the currently used range can be read freely, so read requests for that range will always provide the stored word value and be responded with success. Attempts to read outside the current used range will be responded will failure and trigger a hardware error.

Cartridge program ROMs are always read-only memory so any write operation will be responded with failure and trigger a hardware error.

3.3 Internal variables

The input controller 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, boolean, etc) described in part 2 of the specification. Here we will list and detail all internal variables.

Cartridge connected	**Initial value:** (*)
Format: Boolean	**Valid range:** True / False

This value indicates if a cartridge is connected to the console.

(*) Its initial value is determined by the presence or absence of a cartridge in the cartridge slot on console startup. This value cannot change during console operation.

Program ROM size	**Initial value:** (*)
Format: Integer	**Valid range:** From 1 to 134217728 (= 128*1024*1024)

Represents the program ROM's size in 32-bit words for the currently connected cartridge. This value is 0 when no cartridge is present.

(*) Its initial value is determined by the cartridge (if any) present in the cartridge slot on console startup. This value cannot change during console operation.

Number of textures	**Initial value:** (*)
Format: Integer	**Valid range:** From 0 to 256

Represents the number of textures contained in the video ROM for the currently connected cartridge. This value is 0 when cartridge is absent or does not contain a video ROM.

(*) Its initial value is determined by the cartridge (if any) present in the cartridge slot on console startup. This value cannot change during console operation.

Number of sounds	**Initial value:** (*)
Format: Integer	**Valid range:** From 0 to 1024

Represents the number of sounds contained in the audio ROM for the currently connected cartridge. This value is 0 when cartridge is absent or does not contain an audio ROM.

(*) Its initial value is determined by the cartridge (if any) present in the cartridge slot on console startup. This value cannot change during console operation.

3.4 Control ports

This section details the set of control ports exposed by the cartridge controller 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
500h	00h	Cartridge Connected	Read Only
501h	01h	Program ROM Size	Read Only
502h	02h	Number Of Textures	Read Only
503h	03h	Number Of Sounds	Read Only

3.4.1 Behavior on port read/write requests

Unlike other chips, cartridge controller ports can actually be modeled as read-only registers. The effect of reading these ports is only obtaining their related value, as detailed in this section.

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 success on read requests, and failure on write requests. In this last case the cartridge controller will perform no further actions and the CPU will trigger a HW error.

Cartridge Connected port

On read requests:
The cartridge controller will provide the current value of the internal variable "Cartridge connected".

Program ROM Size port

On read requests:
The cartridge controller will provide the current value of the internal variable "Program ROM size".

Number Of Textures port

On read requests:
The cartridge controller will provide the current value of the internal variable "Number of textures".

Number Of Sounds port

On read requests:
The cartridge controller will provide the current value of the internal variable "Number of sounds".

3.5 Responses to control signals

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

Reset signal:

- If the console is powering on with no cartridge present (so, no cartridge event has been processed yet), the cartridge controller will perform the same processing as when the cartridge is removed.

Frame and Cycle signals:

- The cartridge controller does not need to react to these signals, unless specific implementation details require it.

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

When the cartridge is removed:

- The cartridge controller will its “Cartridge connected” variable to False.
- Internal variables “Program ROM size”, “Number of textures” and “Number of sounds” will all be set to 0.
- All of cartridge controller’s memory addresses will be unassigned.

When a new cartridge is connected:

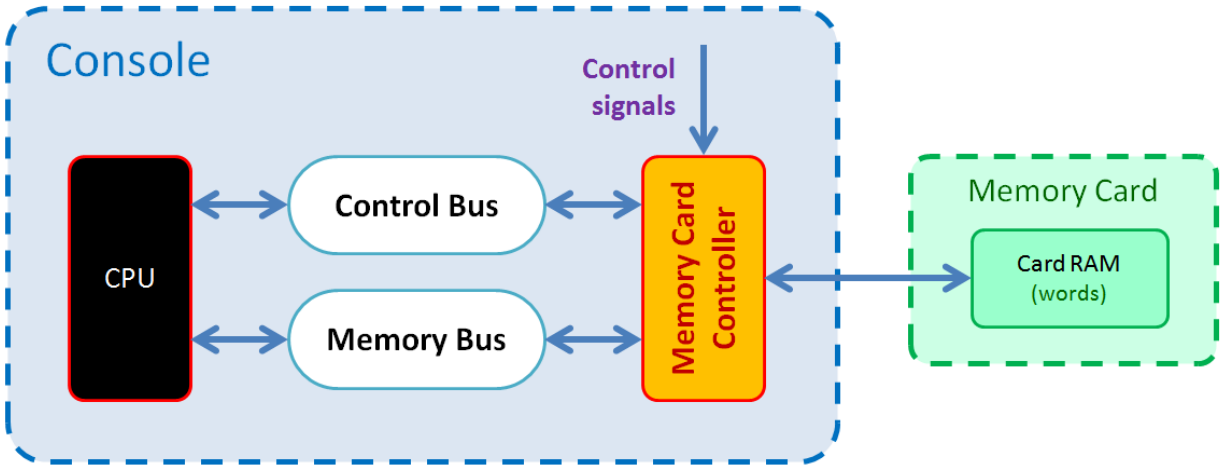
- The cartridge controller will set its “Cartridge connected” variable to True.
- The cartridge controller will perform the connection process described in section 3.2.1 to assign memory addresses to its program ROM and gain access to its contained words.
- Internal variable “Program ROM size” will be set to the number of words contained in the connected cartridge’s program ROM.
- If the connected cartridge contains a video ROM, the cartridge controller will set its internal variable “Number of textures” to its number of contained textures.
- If the connected cartridge contains an audio ROM, the cartridge controller will set its internal variable “Number of sounds” to its number of contained sounds.

4 Memory card controller

The memory card controller is the chip in charge of controlling the console's memory card slot. Same as cartridges, only 1 card can be present. Memory for all cards is the same size and only their stored content will vary. However, unlike cartridges, memory cards can be connected and removed at any moment.

4.1 External connections

Since the memory card controller is just one of the chips forming the console, it cannot operate in isolation. This figure shows all of communications its with other components. As mentioned, the connected memory card (if any) could be a diferent one every time.



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

4.1.1 Control signals

As all console components, the memory card controller receives the signals for reset, new frame and new cycle. The responses to those signals are detailed in section 4.5 of this document.

4.1.2 Control Bus

The memory card controller 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 memory card controller. The list of memory card controller ports as well as their properties will be detailed in later sections.

4.1.3 Memory Bus

The memory card controller is connected as slave device to the Memory Bus, with device ID = 3. This allows the bus master (the CPU) to request read or write operations on the memory addresses exposed by the memory card. The range and properties of card memory addresses will be discussed in later sections.

4.1.4 Memory card

The memory card controller manages the memory card slot. This slot can be queried by the CPU to determine if it currently has a memory card connected to it. When a card is present, the memory card controller will be able to read and modify its contents.

The memory card slot does not feature a blocking mechanism like the cartridge slot. For this reason, programs should check if there is a memory card every time before starting to read or write to it. Any attempt to access the card when none is present will result in a hardware error and stop the program.

4.2 Card memory

The console does not itself contain any permanent storage usable by programs. To save and retrieve data between sessions it needs to write and read words in the card by connecting to its RAM memory. A RAM memory is a read and write memory region that contains a sequence of 32-bit words.

The memory card controller is connected to the memory bus to allow any connected card to expose the contents of its RAM to the CPU. The memory card controller uses device ID = 3 within the memory bus. A memory card RAM contains exactly (256 x 1024) words, so the range of addresses for card memory will be:

Internal addresses → From 0 to 3FFFh (= 256 x 1024 – 1)
External addresses → From 30000000h to 30003FFFh

4.2.1 Connection to card RAM

When a memory card is connected the memory card controller assigns all of its (256 x 1024) internal memory addresses to the card's RAM words in order, starting from 0.

This connection process happens whenever a new memory card is inserted. This can happen at any time, so the memory card controller has to perform connection at that moment. If a card is connected while the console is off, the implementation may choose to delay this connection process until next console power on.

It is up to the implementation to decide how to establish a connection to the card RAM and how to access its words. Card accesses could potentially happen every cycle, so a possible option if the implementation is not fast enough is to work with a copy in system RAM and only write changes to the actual card in certain moments (for example, using the new frame signal).

4.2.2 Behavior on memory read/write requests

Any memory address within the stated range can be read and written freely, so any requests for that range will always provide the stored word value and be responded with success. Attempts to read or write outside the current used range will be responded with failure and trigger a hardware error.

4.2.3 Card connection events

When a card is connected or removed, the memory card controller does not provide any signal to report this event on the fly. Still, the console can know about this when, after a card is connected or removed, the "Card connected" variable has changed its value. Still, implementations can internally work using these events if needed.

4.3 Internal variables

The memory card controller features a single variable that stores its internal state. This variable is stored as a 32-bit value, and it is interpreted using the same data formats (integer, boolean, etc) described in part 2 of the specification. Here we will list and detail this internal variable.

Card connected	**Initial value:** (*)
Format: Boolean	**Valid range:** True / False

This value indicates if a memory card is currently connected to the console.

(*) Its initial value is determined by the presence or absence of a card in the memory card slot on console startup.

4.4 Control ports

This section details the set of control ports exposed by the memory controller via its connection to the CPU control bus as a slave device. The single exposed port, along with its basic properties, is listed in the following table:

List of exposed control ports			
External address	Internal address	Port name	R/W access
600h	00h	Card Connected	Read Only

4.4.1 Behavior on port read/write requests

Unlike other chips, the memory card controller port can actually be modeled as a read-only register. The effect of reading this ports is only obtaining its related value, as detailed in this section.

Card Connected port

On read requests:
The memory card controller will provide the current value of the internal variable “Card connected”. A success response will be provided to the control bus.

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

4.5 Responses to control signals

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

Reset signal:

- If the console is powering on with no card present (so, no card event has been processed yet), the memory card controller will perform the same processing as when the memory card is removed.

Frame and Cycle signals:

- The memory card controller does not need to react to these signals, unless specific implementation details require it.

In addition to reacting to control signals, the memory card controller will also need to perform the following processings in response to console-level events:

When a memory card is connected:

- The memory card controller will set its “Card connected” variable to True.
- The memory card controller will perform the connection process described in section 4.2.1 to assign memory addresses to the card RAM and gain access to its contained words.

When the memory card is removed:

- The memory card controller will set its “Card connected” variable to False.
- All of memory card controller’s memory addresses will be unassigned.

(End of part 6)