« Architecture of Consoles

Sega Master System Architecture

A practical analysis by Rodrigo Copetti

Classic edition - Last updated: May 18, 2025

Languages available: 🇬🇧 - English, 🇵🇱 - Polski, 🇨🇳 - 简体字,
👋 - Add translation

About this edition

The ‘classic’ edition is an alternative version to the ‘modern’ counterpart. It doesn’t require Javascript, state-of-the-art CSS or convoluted HTML to work, which makes it ideal for readers who use accessibility tools or legacy internet browsers. On the other hand, paper/eBook users can now check the book editions.

This edition is identical content-wise. However, interactive widgets have been simplified to work with pure HTML, though these will offer an link to the original article in case the reader wants to try the ‘full version’.

As always, this article is available on Github to enable readers to report mistakes or propose changes. There’s also a supporting reading list available to help understand the series. The author also accepts donations and translations to help improve the quality of current articles and upcoming ones.

Table of Contents

---

1. Supporting imagery

1.1 Models

Figure 1.1.1: The Sega Master System.
Released on 20/10/1985 in Japan, 09/1986 in America and 06/1987 in Europe

1.2 Motherboard

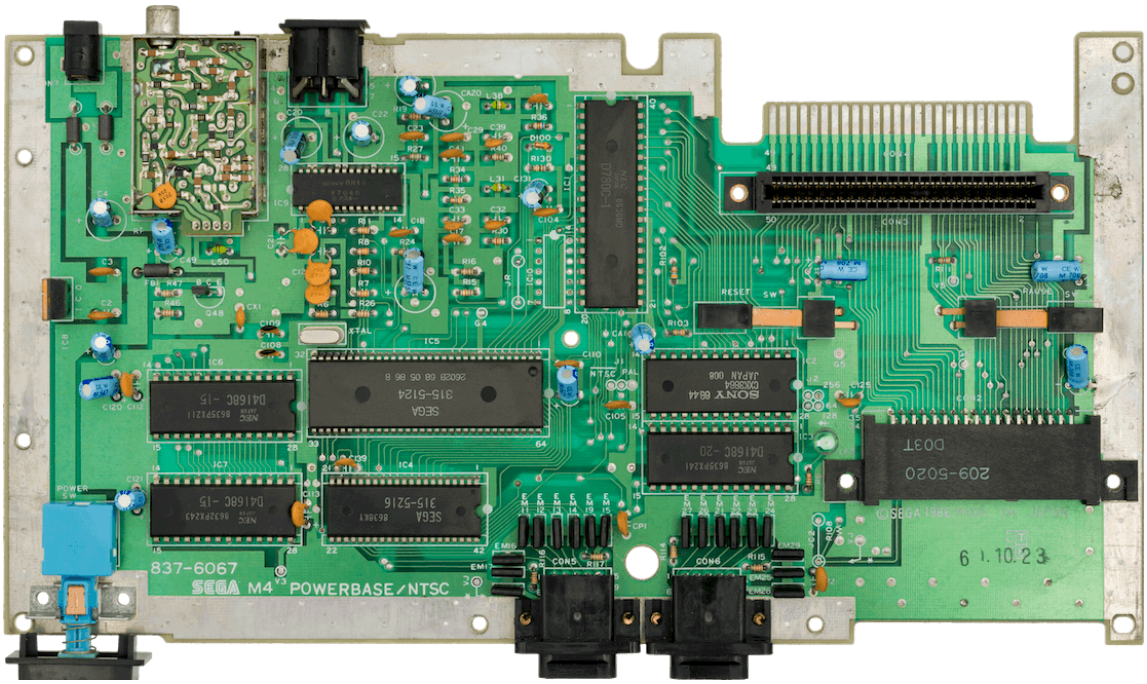


Figure 1.2.1: Motherboard

The sound chip is embedded in the video display processor.

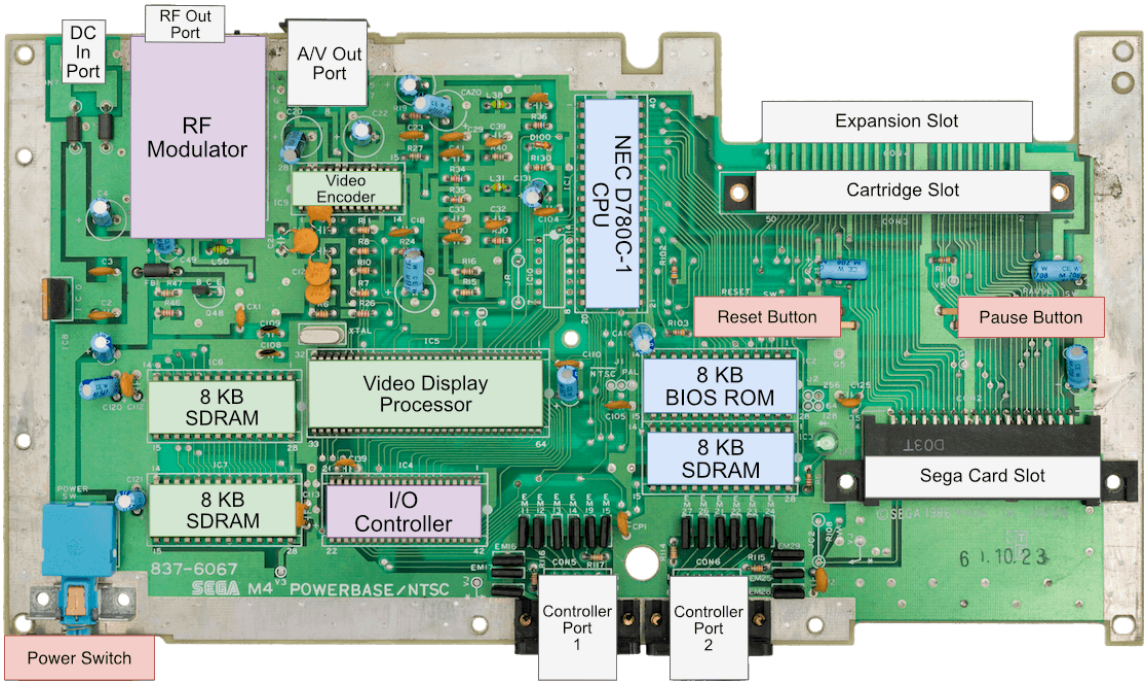


Figure 1.2.2: Motherboard with important parts labelled

1.3 Diagram

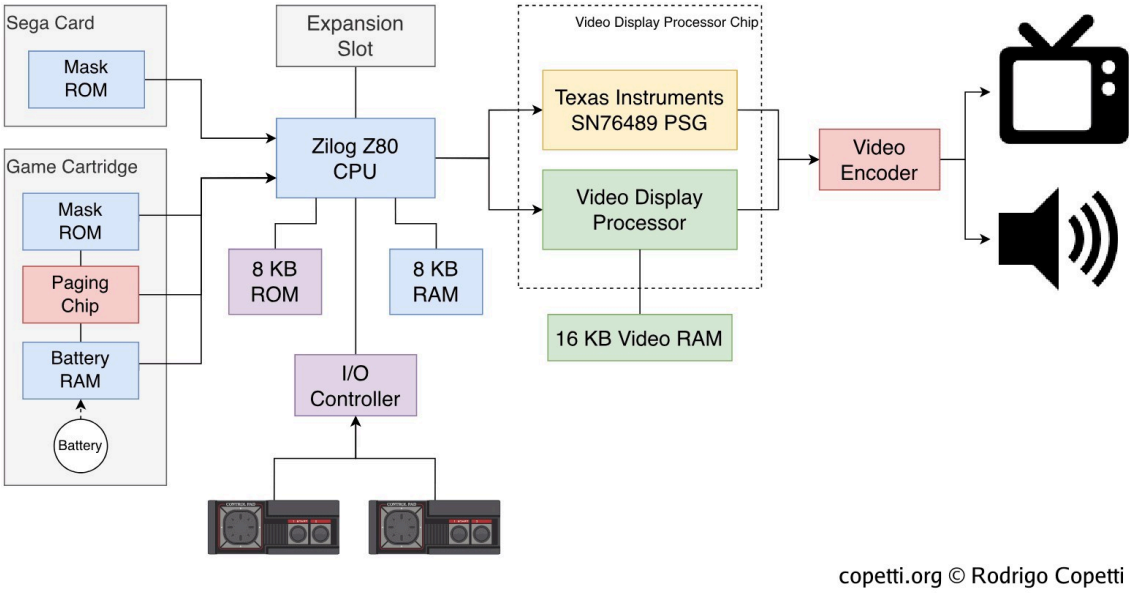


Figure 1.3.1: Main architecture diagram

---

2. A quick introduction

The Master System comes from a long line of succession. What began as a collection of off-the-shelf components has now acquired a new identity thanks to Sega's engineering.

---

3. Models and variants

I was a bit confused at first when reading about the different models that Sega eventually shipped. So, here is a summary of the main models to clarify any potential confusion:

- **Sega Mark III**: The first console featuring this architecture, released exclusively in Japan.
- **Sega Master System** (Europe and America): A rebranded Mark III with a new case, a BIOS ROM chip, and a different cartridge slot.
- **Sega Master System** (Japan): A European/American Master System with the Mark III's cartridge slot, a new FM chip, and a jack port for '3D glasses'. However, it lacks the RESET button.

From now on, I'll use the term 'Master System' or 'SMS' to collectively refer to these models, except when discussing features exclusive to a specific variant.

---

4. CPU

Sega opted for a fully-fledged **Zilog Z80** CPU running at **~3.58 MHz**, a popular choice among computer manufacturers such as Sinclair, Amstrad, and Tandy.

Figure 4.0.2: The American Tandy TRS-80 (1977). I found it in the Computer History Museum (Mountain View, California), during my visit in June 2019.

Figure 4.0.3: The British Sinclair ZX80 (1980), it is way smaller than it looks! This is part of Christopher Rivett's collection [1].

Figure 4.0.3: Popular computers across the world carrying a Z80 CPU.

The Z80 CPU has an interesting background, as it was designed by none other than the authors of the Intel 8080 (Federico Faggin and Masatoshi Shima), who became disenchanted with Intel's direction and decided to start their own silicon company, Zilog, in 1974.

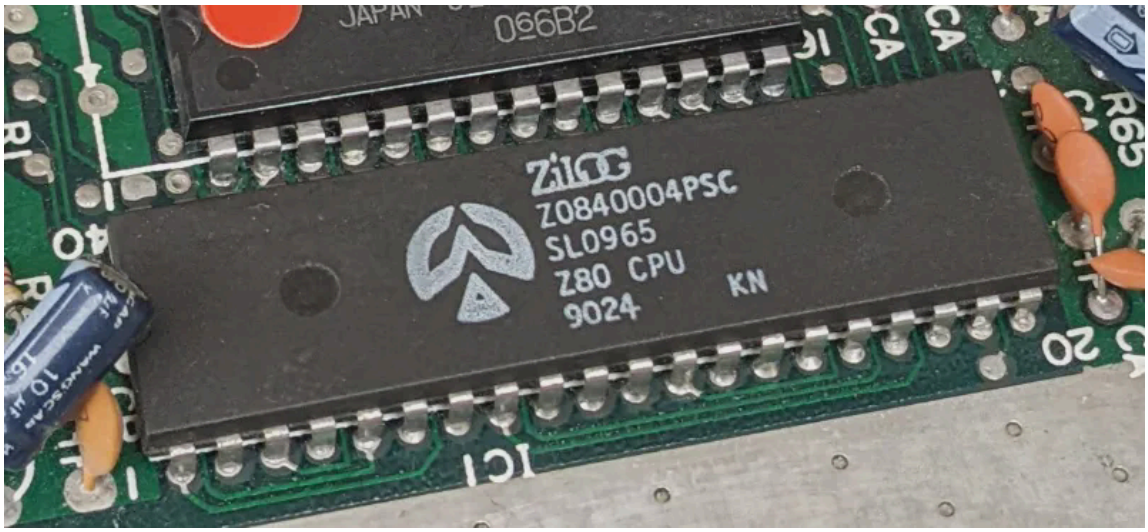


Figure 4.0.4: The Z80 chip on the Master System's motherboard

Their debut product can be considered an unofficial successor to the Intel 8080, now featuring:

- The **Z80 ISA**: An instruction set compatible with the Intel 8080 but expanded with lots more instructions. It handles **8-bit** words.
- An **8-bit data bus**, ideal for moving 8-bit data around. Larger values will consume extra CPU cycles.
- **Fourteen 8-bit general-purpose registers** [2]: This is quite a lot, considering the Intel 8080 features half and the MOS 6502 only three. However, the Z80's register file exhibits some caveats (or advantages, depending on how you see it):
  - Only **seven registers are accessible at a time**, the other seven are called 'Alternative Registers' and must be swapped with the first set to be able to access them. This aligns with the principle of bank switching. Also, the Z80 provides specialised instructions like EX and EXX to transfer the contents between each set.
  - Within each set, six 8-bit registers may also be paired together to provide up to **three 16-bit registers**, allowing the manipulation of larger values.
- A **16-bit address bus**, its consequences are explained in the next section.
- A **4-bit ALU**: This may be a bit shocking, but it simply means that operations on 8-bit values take twice as many cycles to compute.

The motherboard picture at the start of the article shows an NEC D780C-1 CPU, which is simply SEGA second-sourcing the chip to different manufacturers. Other

revisions even included the chip manufactured by Zilog. But for this article, it doesn't matter who fabricated the CPU, as the internal features remain the same.

4.1 Relative comparison

Notice how the 6502 CPU (featured in the NES) runs at a mere ~2 MHz, which is about half the speed of the Master System's chip. Combined with the Z80's larger register file, one might assume the Master System would outperform the NES without a doubt.

Conversely, if we dig deeper, we see that the 6502 houses a larger (8-bit) ALU. Thus, arithmetic operations that may only take two cycles in the 6502, **will consume four** on the Z80. Ultimately, this shows that relative qualities like CPU clock speed or register file size, when analysed in isolation, may be deceiving. Both the Z80 and 6502 excel and struggle at different tasks; it all comes down to the skills of the programmer.

4.2 Memory available

As mentioned earlier, the Z80 has a 16-bit address bus, so the CPU can find up to **64 KB worth of memory**. Now, in the Master System's memory map, you'll find **8 KB of RAM** for general-purpose use [3], this is mirrored in another 8 KB block. Finally, **up to 48 KB of game ROM** are mapped as well.

4.3 Accessing the rest of the components

As you may have read in the previous paragraph, only the main RAM and some cartridge ROM are found in the address space. So, how does the program access other components? Well, unlike Nintendo's Famicom/NES, not all the hardware in the Master System is mapped to memory locations. Instead, certain peripherals are accessed through the **I/O space**.

This is because the Z80 family contains an interesting feature called **I/O ports**, which enable the CPU to communicate with other hardware without exhausting memory addresses. To achieve this, there is a separate address space for 'I/O devices' called **ports**, and both share the same data and address bus. The difference, however, is that ports are read from and written to using `IN` and `OUT` instructions, respectively, as opposed to the traditional load/store instruction (`LD`).

When an `IN` or `OUT` instruction is executed, the Z80 sets up the address lines to point to the peripheral (which could be, for instance, a keyboard), flags its `IORQ` pin to indicate that an I/O request has been initiated, and finally flags the `RD` pin or the `WR` pin depending on whether it's an `IN` or `OUT` instruction, respectively. Consequently, the addressed peripheral must manually monitor the address bus and the I/O pins and perform the required operation. In the case of an `IN` instruction, the CPU stores the received value in a predefined register.

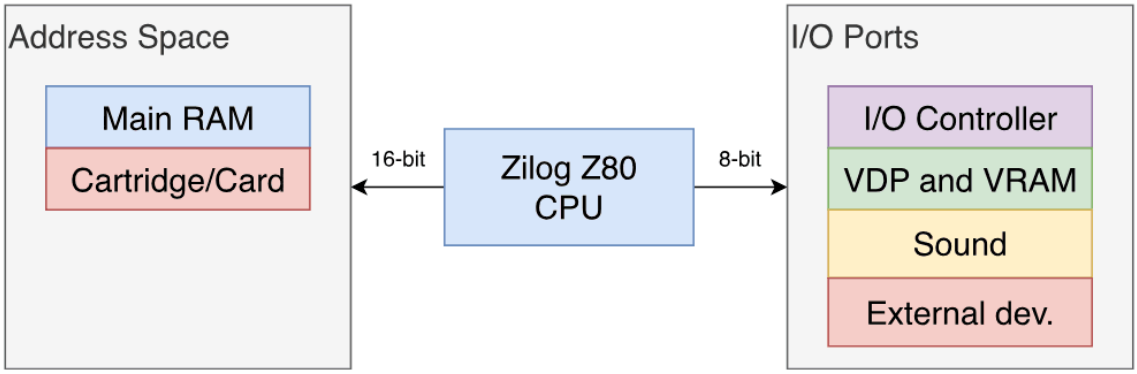


Figure 4.3.1: SMS' Addressing layout.

The way SEGA interconnected the CPU with the rest of the components enables it not only to access values but also to expose or conceal certain components from the memory map.

Interestingly, the Game Boy housed a Z80 'variant' that completely omitted the I/O ports. Thus, it had to fit everything into the memory map.

4.4 Backwards compatibility

The architecture of this console is very similar to its predecessor, the **Sega SG-1000**. This allowed the Master System to gain backwards compatibility with the SG-1000. Although, this only applies to the Japanese variant, as others feature a different cartridge slot.

---

5. Graphics

The drawings on the screen are produced by a proprietary chip called **Video Display Processor** or 'VDP'. Internally, it shares the same design as the Texas Instrument TMS9918 (used in the SG-1000) [4], though it has been enhanced with features which we will discuss in the following sections.

5.1 Organising the content

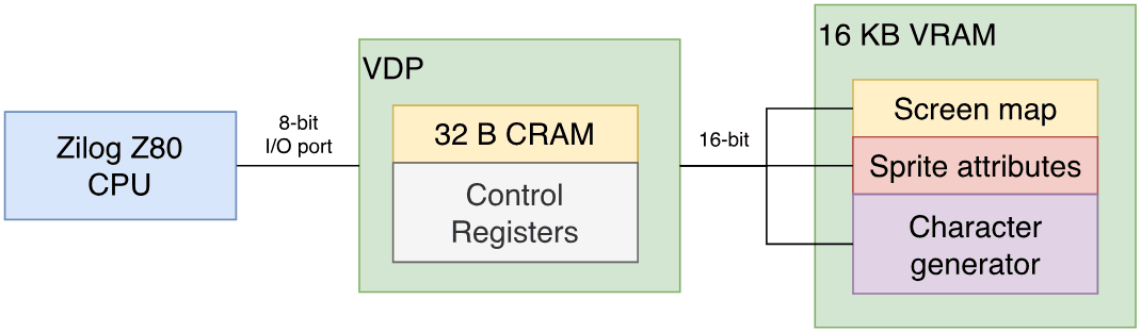


Figure 5.1.1: Memory architecture of the VDP.

Connected to the VDP are **16 KB of VRAM**, which only the VDP can access using a **16-bit data bus** (Sega modified the original design to access two memory chips with 8-bit buses simultaneously [5]). If you look at the motherboard picture again, you'll notice that both RAM and VRAM chips are roughly the same, except that VRAM uses the chip model ending in '20', which offer lower latency [6].

For the Master System, VRAM houses everything the VDP requires for rendering (except Colour RAM). The CPU fills VRAM by writing to specific VDP registers, which in turn forward the values to VRAM. Since the VDP is accessed through I/O ports, the CPU must use `IN` and `OUT` instructions.

5.2 Constructing the frame

The VDP renders frames with a resolution of **up to 256x192 pixels**. Later revisions added support for 256x224 px and 256x240 px. However, to maintain compatibility with all models, developers adhered to the standard resolution. This chip has the same *modus operandi* as Nintendo's PPU; in other words, graphics are rendered on the spot.

Furthermore, the VDP has four different modes of operation. These alter the characteristics of the frame (colour depth and resolution):

- **Mode 0 to III**: Inherited from the TMS9918 found on the SG-1000. They are included for backwards compatibility, although any SMS game can use them.

- **Mode IV**: The native mode of the Master System, which enables access to all the state-of-the-art features of the VDP. For this analysis, we will focus on this mode!

Let us now examine how a frame is drawn step by step. For this, I'll borrow *Sonic The Hedgehog*'s assets. Also, to make explanations easier, this analysis will focus on the standard memory layout that Sega suggests for organising the graphics content (just remember that the VDP is very flexible, allowing games to optimise the layout).

5.2.1 Tiles

Figure 5.2.2: All tiles.

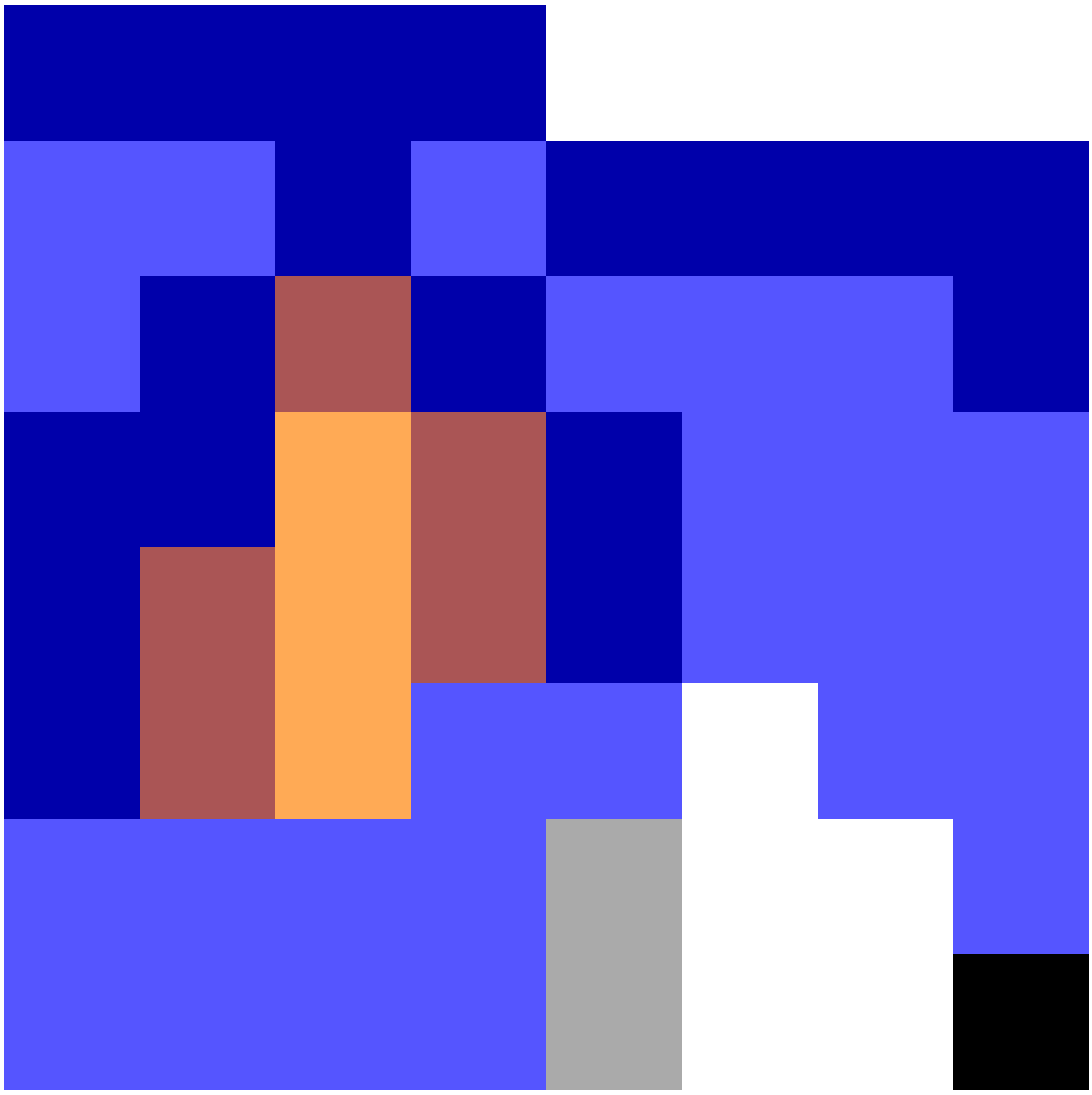


Figure 5.2.3: A single tile.

Figure group: Tiles Found in VRAM.

Mode IV is based on the **tile system**. To recall previous explanations about tile engines, tiles are just **8x8 pixel bitmaps** that the renderer retrieves to draw the game's graphics. In the case of the VDP, the frame consists of two planes: the background layer and the sprite layer.

Inside VRAM, there is an area dedicated to tiles called **Character generator** (Sega refers to tiles as 'Characters'), and it's set to be **14 KB long**. Each tile occupies 32 bytes, so we can store up to 448 tiles.

Each tile contains 64 pixels, and the VDP rules that each pixel must use 4 bits, allowing **16 colours can be chosen**. These bits reference a single entry in **Colour RAM** or 'CRAM', which is located within the VDP and stores the colour

palettes. Colour palette systems reduce the size of tiles in memory and enable programmers to alternate their colours without storing multiple copies.

Colour RAM stores **two palettes of 16 colours each**. Each entry is 6 bits wide, with each 2-bit set defining one colour from the RGB model. This provides a total of 64 colours to choose from.

5.2.2 Background Layer

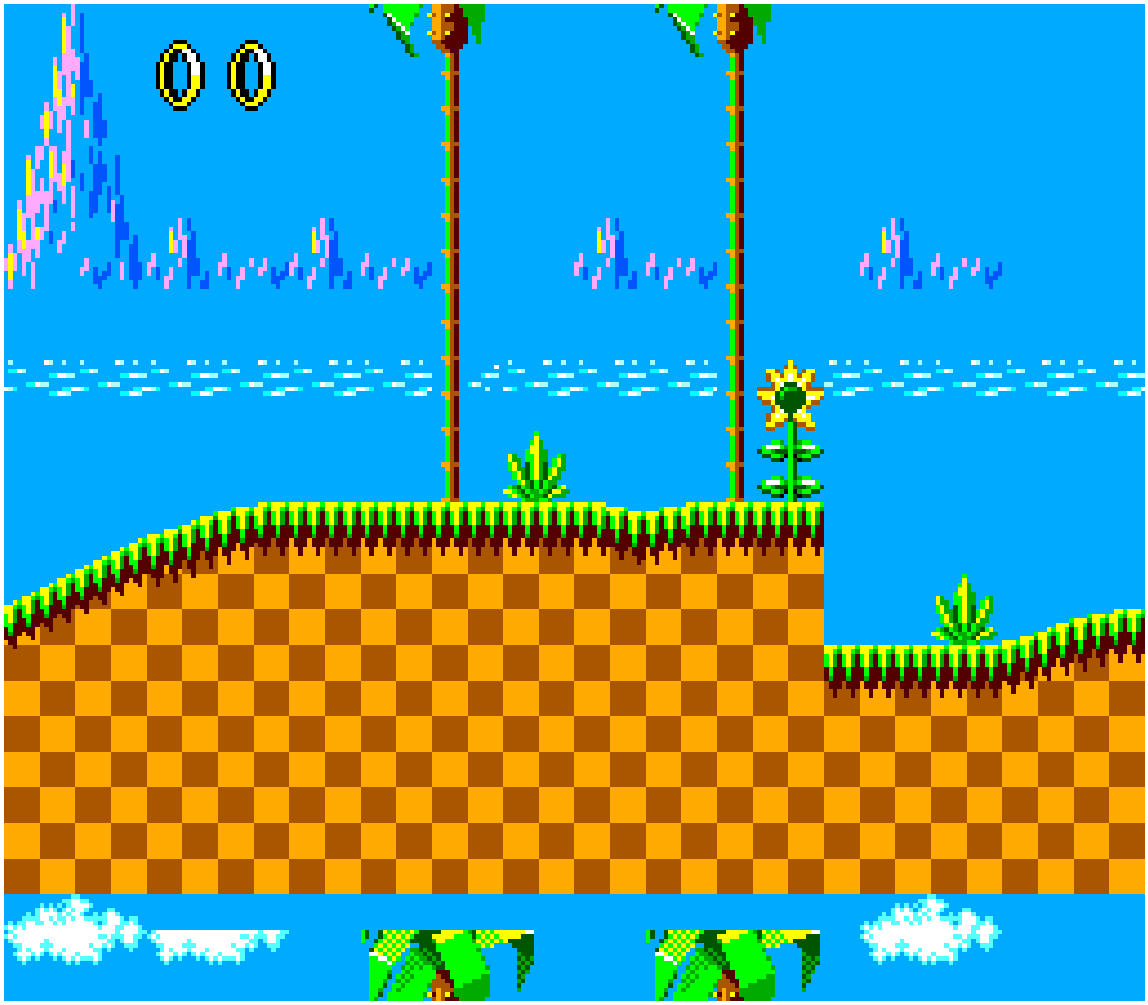


Figure 5.2.4: Allocated Screen map.

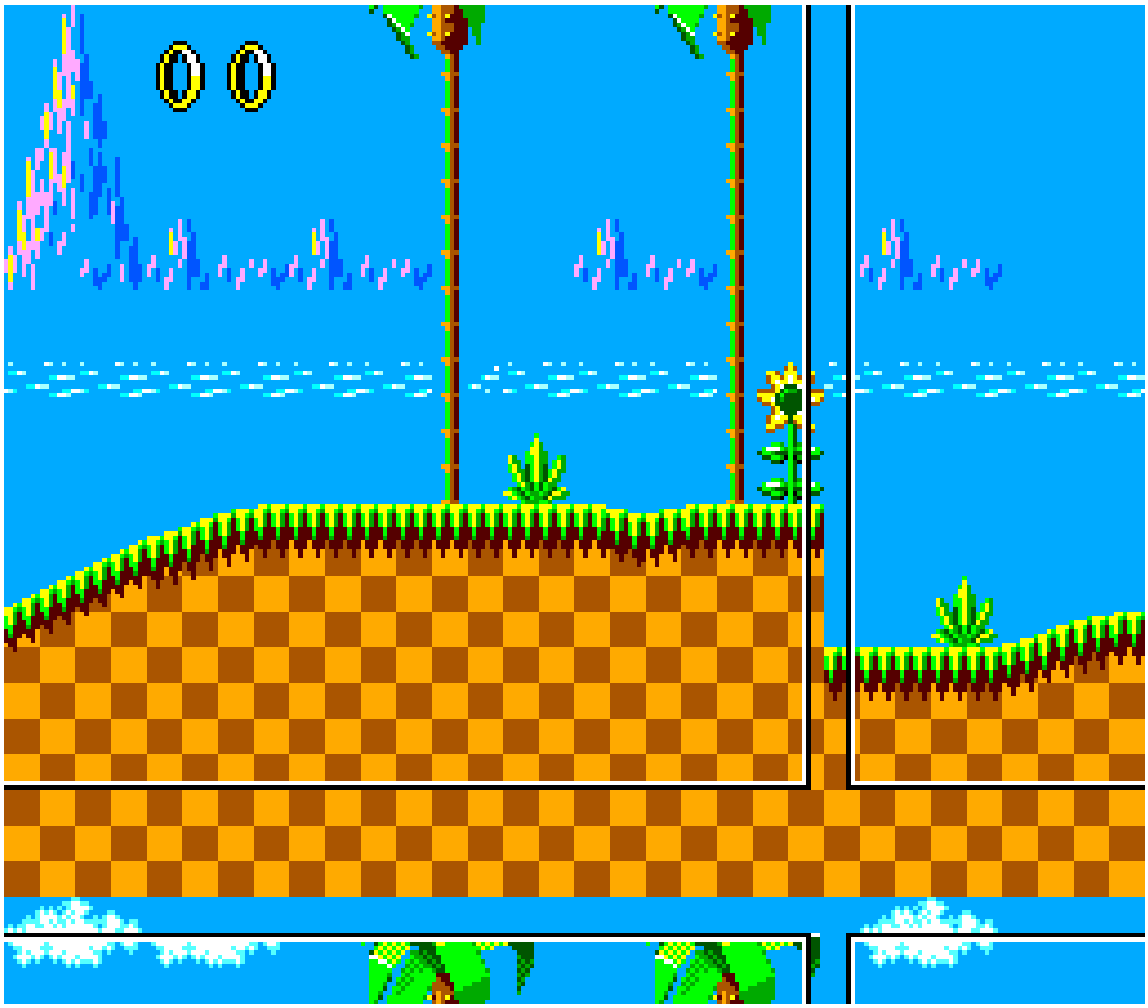


Figure 5.2.5: Allocated Screen map with selected area marked.

The background layer is a large plane where static tiles are drawn. To place something here, there is another area in VRAM called **Screen map**, which takes 1.75 KB.

This allows developers to build a layer of 896 tiles (32x28 tiles) [7], but if we do the math, we see that this layer is larger than the display resolution of the console. The reality is, only 768 tiles (32x24 tiles) are visible, so the visible area is manually selected at the programmer's will. Also, by slowly alternating the X and Y coordinates of the selected area, a **scrolling effect** is achieved.

Each entry in the map is 2 bytes wide (matching the width of the VDP's data bus) and contains the address of the tile in the Character generator, along with the following attributes:

- **Horizontal and Vertical flip**.
- The **priority bit** (determines whether to draw some or all the tile in front of sprites).
- The **colour palette** used.

Curiously enough, there are three unused bits in the entry that the game can use for other purposes (e.g., extra flags to assist the game engine).

5.2.3 Sprites

Figure 5.2.6: Rendered Sprite layer.

Sprites are essentially tiles that can move freely. The VDP can raster **up to 64 sprites** using a single tile (8x8 px) or two tiles stacked vertically (8x16 px).

The **Sprite Attribute Table** is a 256-byte area in VRAM that contains an array of all the defined sprites. Its entries are similar to those of the background layer, except that each sprite contains two additional values representing the X/Y coordinates.

The VDP is limited to **a maximum of eight sprites per horizontal scan-line** [8]. Also, if multiple sprites overlap, the first one in the list is displayed.

5.2.4 Result

Figure 5.2.7: Tada!

The VDP automatically blends the two layers to form the final frame. The rendering process is performed scan-line by scan-line, so the VDP doesn't really know how the frame will look; that's only visible to the user once the picture is constructed on the TV.

If you examine the example image, you may notice a vertical column at the left side of the frame. This occurs because the screen map is tall enough to provide vertical scrolling without producing artefacts, **but not wide enough for horizontal scrolling**. To address this, the VDP can **mask** the left-most side with an 8 px column to prevent intermediate tiles from being displayed.

To update the graphics for the next frame without disrupting the image currently being displayed, the VDP sends two types of **interrupts** to the CPU. One notifies that the CRT TV has finished beaming a chosen number of scan-lines (called **horizontal interrupt**), while the other signals that the CRT has completed drawing the final scan-line (called **vertical interrupt**), indicating the frame is finished. During these events, the CRT's beam is repositioning to draw the next scan-line (**blanking interval**), so any alteration of the VDP's state won't tear the image. Horizontal blanking has a shorter time-frame than vertical blanking, yet it

still allows to change, let's say, the colour palette. This still can produce certain effects.

5.3 Secrets and limitation

At first glance, the VDP may seem like another chip with minimal functionality that we now take for granted. Although, it still managed to draw significant attention away from Nintendo's offering at the time. Why was that?

5.3.1 Collision detection

First of all, the VDP could **tell if two sprites were colliding**. This was done by checking its `status` register [9]. While it could not identify which sprites in particular were colliding, developers could 'triangulate' the location by reading other registers as well, such as the `scan-line counter`.

This feature was not new, actually, as the TMS9918 also included it. Thus, the SG-1000 also had collision detection.

5.3.2 The need for modularity

When I previously analysed the design of Nintendo's PPU, I emphasised its internal memory architecture. While limited, some constraints were somewhat beneficial as they enabled the system to be expanded with extra hardware included in the game cartridge, keeping the costs down in the process.

The VDP doesn't take advantage of this modular approach. Instead, Sega implemented a different solution that in turn saves cartridge costs. The smaller background layer and horizontal interrupts are examples of this.

5.3.3 3D glasses

Figure 5.3.1: Sega 3-D glasses [10]. The American variant connected through the card port.

It turns out Sega also shipped **‘3D glasses’** as an official accessory! The glasses worked in sync with the CRT. During gameplay, the game alternates the position of objects between frames. Each lens contains an LCD screen that turns black to block your vision. The correct combination of graphics flickering and alternating shutters eventually creates a stereoscopic image in your head, resulting in a ‘3D’ effect.

The shutters are controlled via several memory addresses, but none of them indicate whether the glasses are actually plugged in. Therefore, games that support this accessory include a settings option to manually activate this feature.

The LCD controllers are interfaced with a jack cable, which plugs into the console. The European and American versions did not include the jack input, so they rely on the card port to connect an adaptor (we’ll see more about the card slot later on).

5.4 Video Output

The video-out connector of this system is *incredibly* versatile. It exposes both **composite** and **RGB** signals, which can be considered the two ‘extremes’ of video quality.

The downside, however, is that it doesn’t carry ‘composite sync’, so using the RGB output requires capturing the sync signal from composite, and its quality isn’t optimal.

6. Audio

The audio capabilities of this console are pretty much aligned with other 80s equipment. Inside the VDP chip, we find a slightly-customised version of the **Texas Instruments SN76489** [11], which is a **Programmable Sound Generator** or ‘PSG’. This is the same type used in the NES/Famicom, albeit having different functions.

6.1 Functionality

A PSG can only synthesise a limited set of waveforms, with each channel allocated to a single waveform. I previously introduced some PSGs in the NES and the Game Boy article, if you are interested in this type of sound synthesis.

In the SMS, the PSG is programmed by altering its set of registers through the aforementioned I/O ports.

Let us now examine each type of waveform the SN76489 can synthesise:

6.1.1 Pulse

0:00

Figure 6.1.1: Oscilloscope display of Sonic The Hedgehog (1991), showing the pulse channels.

Pulse/Tone waves produce that iconic sound of the 8-bit generation. The sound wave is generated by latching up the voltage, holding it steady, and then dropping it. Repeating this at a constant rate produces a tone.

The period of the wave defines the frequency of the sound (musical note), while its duty cycle affects the timbre.

The SN76489, in particular, can produce **three pulse waves at the same time**. It also exposes a 10-bit counter on each channel that is used internally to latch the output, resulting in pulse waves with programmable frequencies.

6.1.2 Noise

0:00

Figure 6.1.2: Oscilloscope display of Sonic The Hedgehog (1991), showing the noise channel.

Noise is a type of signal associated with interference. When outputted to a speaker, it sounds like static.

The SN76489 contains a Linear Feedback Shift Register (LFSR) which is fed with some input and cycled through to generate a pseudo-random signal. It can alternate between **white noise** and **periodic noise**.

The oscillator offers three frequencies to choose from, which alter the pitch of the noise [12]. There's also a fourth option where the output of the third pulse channel modulates the noise channel. Some games used this option, combined with the periodic mode, to produce bass-like sounds (similar to a pulse at four octaves lower).

Overall, games typically use the noise channel for **percussion or sound effects**.

6.1.3 Mixed

Figure 6.1.3: Oscilloscope display of Sonic The Hedgehog (1991), showing all audio channels.

So far we have discussed what each channel does individually. In practice, the TV receives a mono signal with all the channels mixed by the PSG.

Finally, the chip also contains programmable attenuators that lower the decibels of each channel, effectively acting as a **volume control**.

6.2 Secrets and limitations

Just like the VDP, the PSG is a no-brainer, but it does hide some interesting functionality:

6.2.1 FM Expansion

Interactive player available in the modern edition

Figure 6.2.1: Audio samples
Double Dragon (1987).

The Japanese version of the Master System embedded an extra chip made by Yamaha called **YM2413**. It is drastically different from the PSG, as it uses **frequency modulation** to generate sound. I briefly explained how this works in the Mega Drive article, in case you are interested.

This chip adds **nine audio channels**. Each channel can either select one of the 16 preset instruments, or define a custom one by programming the carrier and modulator. Unfortunately, only one custom instrument is allowed at a time. On the other hand, the custom instrument offers interesting functions, such as ADSR envelope controls and feedback.

The YM2413 also has a second mode of operation called **Rhythm mode**, which instead provides **six channels** supplemented with **five additional channels** for rhythm instruments only.

The final sound output is generated by the YM2413, which mixes its channels with those of the PSG.

The Mark III variant did not include this chip, but FM functionality was available as an expansion unit called the **FM Sound Unit**. The European and American Master Systems, however, had to stick with the PSG, although some third-party modifications eventually appeared.

6.2.2 Emulation accuracy

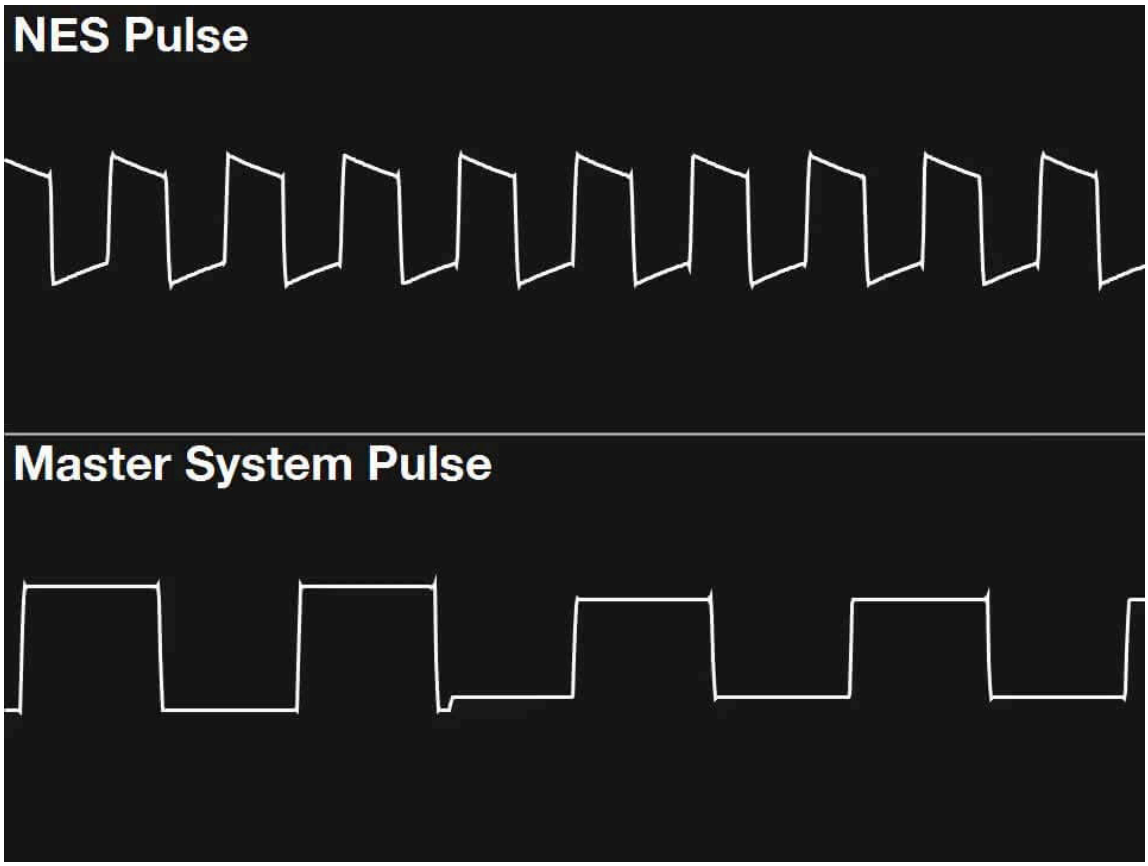


Figure 6.2.2: Pulse waveform comparison using emulators. NES' shows some decay while SGM's is squared-shaped.

While reading through SMS Power (a website that collects extensive technical information about the system), I came across an interesting section called 'The imperfect SN76489' [13], which discusses some discrepancies I encountered while writing the article.

If you revisit the example video for the pulse wave, you will notice that it is visualised as an almost-perfect square wave. The pulse generator works by latching the voltage to generate the tone. However, in electrical circuits, components don't just go from zero to one (or vice-versa) instantly; there is always a transient period, primarily due to the presence of filters in the electrical circuit.

Now, I use emulators to capture separate channels and avoid interference during recordings. Although, emulators do not always account for the transient factor, so their result may be closer to the 'ideal scenario' than the realistic behaviour found in 80s electronics. Take a look at the example image: both samples are taken from emulators, but the NES one potentially exhibits a behaviour closer to the analogue world.

I don’t have the necessary tools right now to confirm whether the SMS should exhibit similar behaviour. However, if it does, this does not imply that what you heard is incorrect - only that it may be at a slightly different volume, which is barely noticeable.

6.2.3 Sample play

Figure 6.2.3: Oscilloscope display of Alex Kidd - The Lost Stars (1986), showing the 1-bit PCM sample.

While the SN76489 lacks a PCM channel for reproducing samples, there are some tricks that can be used to simulate this functionality.

These rely on the pulse channels, it was discovered that if the tone level is fixed at 1, the volume level (which alters the amplitude) conditions the shape of the waveform.

smspower.org describes various designs that enable the playback of 1-bit, 4-bit and 8-bit PCM samples. Although, storage requirements skyrocket with higher sample resolutions and rates, so these techniques are best exploited in homebrew games.

It is worth noting that streaming samples consumes a good amount of CPU cycles, and since there is only one processor in this system, the game may need to be halted for a short period.

---

7. I/O

Like other systems from its generation, the CPU is primarily responsible for handling I/O. In this case, the Z80 processor is unique for its specialised I/O addressing, but CPU cycles are still spent moving bits between components.

On the other side, the SMS uses a dedicated **I/O controller** chip to interface with the joypads and to enable or disable parts of the system, which alters the address map. Furthermore, this controller is essential for supporting the FM expansion, as the FM module exposes ports that conflict with the rest of the system.

7.1 Available interfaces

In addition to the two controller ports, the system contains one proprietary cartridge slot, one ‘Sega Card’ slot, and one expansion slot reserved for ‘future accessories’. The latter was never utilised, except for the FM expansion in the Mark III. Even so, the SMS and Mark III featured different expansion port designs [14].

7.2 Top interruptors

Another speciality about this console is the inclusion of two buttons on the top of its case: PAUSE and RESET. You can already guess what they do!

Figure 7.2.1: Top of the case [15].

Pressing the PAUSE button sends a non-maskable interrupt to the CPU [16]. The interrupt handler is stored within the game itself, meaning it is up to the game to honour the press.

In contrast, and for some strange reason, the RESET button is handled as a keypress on the controller.

---

8. Operating System

There's a small **8 KB BIOS ROM** fitted on the motherboard that gets executed whenever the console is powered on. The program itself does not qualify as an 'Operating System'; it is more accurately described as a **Boot Manager**.

8.1 Medium selector

The main goal of the BIOS is to bootstrap a valid game (from either game slot) in the following order of priority: the Sega Card, the cartridge, and the expansion module.

The boot process works as follows:

1. The console is switched on.
2. The BIOS copies part of its code to main RAM.
    - This is a crucial step, since the program will start manipulating I/O ports, which will eventually disable access to the ROM!
3. Show the splash screen (only in USA/Europe models).
4. Check each slot for a valid game.
    - This is done by talking to the I/O controller chip to activate the required slot. Then, the boot program copies the game header (16 bytes) from each slot to check if the game content is valid (thus, properly inserted). The header must have `TMR SEGA` encoded.
5. Perform the region check.
6. Redirect execution to the game.

8.2 Surprise screen

If any of the checks fail, the console loops indefinitely while showing a screen that prompts the user to insert a valid game.

Figure 8.2.1: USA/Europe error message (after the initial splash).

Figure 8.2.2: Japanese ‘error’ message (enhanced by the FM chip!).

I have to say, in terms of the design of the error screen, the creative differences between regions are quite significant. The first time I heard the colourful Japanese version, I thought it was based on *Electric Light Orchestra* (the band), but it is actually from *Space Harrier* (the game). Also, the perspective effect on the floor is accomplished by altering the colour palettes.

8.3 More regional differences

Because the Japanese variant was backwards compatible with the SG-1000, the header check is replaced with an ‘integrity check’ that instead reads data from the first 256 bytes multiple times to detect if it is garbage.

Furthermore, the Mark III doesn’t have a BIOS, so the slots are activated with hardware switches. Furthermore, the cartridge is the medium with the most priority.

8.4 Updatability and later BIOS chips

The BIOS ROM, by its nature, is **not updatable**. Although, as new console revisions entered the market, it was discovered that Sega also updated the BIOS

program.

Later ROMs even embedded entire games! As a consequence, the chip got bigger and was accompanied by a dedicated mapper.

---

9. Games

To make a long story short, games were written in plain Z80 assembly, that's it. There were no compilers or high-level tools back then, apart from the assembler.

9.1 Medium

Figure 9.1.2: Example of game cartridge [17].

Figure 9.1.3: Example of Sega Card [18].

Figure 9.1.3: The two channels of distribution.

The Master System supports two different media for distributing games:

- The **Cartridge**: The most common one, capable of addressing up to 48 KB of memory. However, by including a mapper, the system could access a wider range and handle additional chips, such as battery-backed RAM for storing saves.
    - Sega offered to developers official mappers called ‘Paging Chips’. The most powerful one could map up to 512 KB of memory.
- The **Sega Card**: Visibly thinner and cheaper to manufacture, but it could only access up to 32 KB of memory. Since SEGA never designed a mapper for this medium, the largest card found on the market contained a 32 KB ROM.

10. Anti-Piracy and Region locking

Unlike Nintendo, Sega did not employ aggressive methods to control the distribution of their games. They did, however, prevent American and European systems from running Japanese games by altering the shape of the cartridge slot and implementing a different ROM header check.

Furthermore, American and European systems were required to include the aforementioned `TMR SEGA` code in their header. I suppose this enabled them to prevent unauthorised distribution by leveraging trademark laws.

---

11. That's all folks

Having previously written the Nintendo DS article really puts into perspective how complicated technology has become. The Master System is very straightforward by comparison, even after some 'technical nitpicking' I may have expressed here and there.

Anyway, I hope this article provided you with a comprehensive overview of the state of technology of the early-to-mid-80s. I also want to thank the smspower.org community and the /r/Emulation discord for reading the first draft and pointing out *lots* of corrections and suggestions.

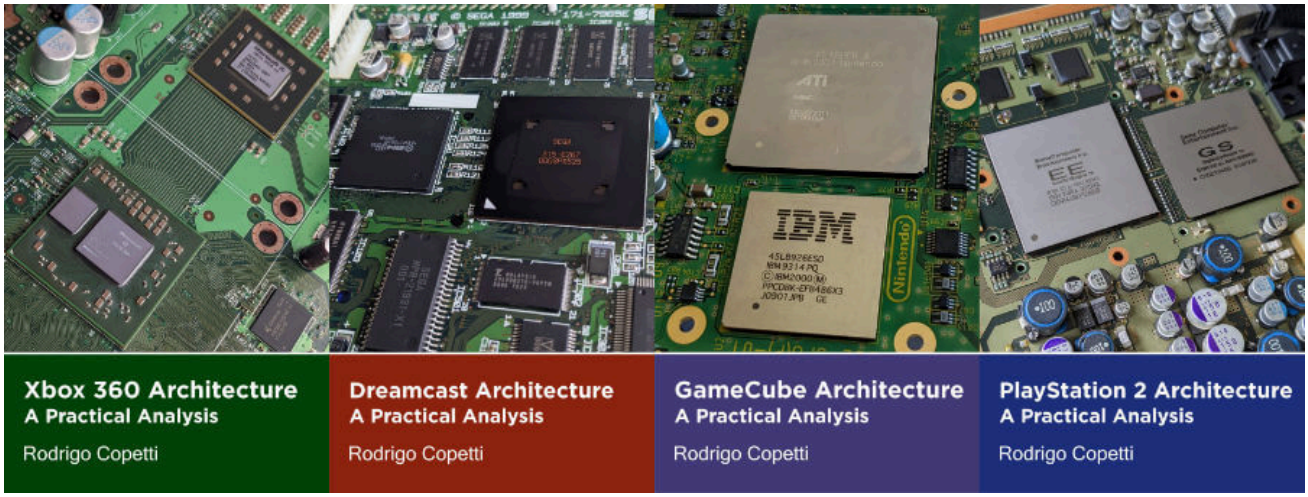
Until next time!
Rodrigo

Dedicated to the memory of Jacinto 'Pocho' Fornasier.

Contributing

This article is part of the Architecture of Consoles series. If you found it interesting then please consider donating. Your contribution will be used to fund the purchase of tools and resources that will help me to improve the quality of existing articles and upcoming ones.

You can also buy the book editions in English. I treat profits as donations.



Big thanks to the following people for their donation:

- Alberto Cordeddu
- Alexander Perepechko
- Andrew Woods
- Colin Szechy
- David Bradbury
- David Sawatzke
- Eric Haskins
- Guillermo Angeris
- Josh Enders
- Sanqui
- Sébastien Lethuaire

Alternatively, you can help out by suggesting changes and/or adding translations.

Copyright and permissions

Article information and referencing

For any referencing style, you can use the following information:

- **Title of article**: Sega Master System Architecture - A Practical Analysis
- **Author**: Rodrigo Copetti
- **URL**: https://classic.copetti.org/writings/consoles/master-system/
- **Date of publication**: October 12, 2020
- **Last modified**: May 18, 2025

For instance, to use with BibTeX:

```
@misc{copetti-mastersystem,
    url =
{https://classic.copetti.org/writings/consoles/master-
system/},
    title = {Sega Master System Architecture - A Practical
Analysis},
    author = {Rodrigo Copetti},
    year = {2020}
}
```

or a IEEE style citation:

```
[1]R. Copetti, "Sega Master System Architecture - A
Practical Analysis", Copetti.org, 2020. [Online].
Available:
https://classic.copetti.org/writings/consoles/master-
system/. [Accessed: day- month- year].
```

Special use in multimedia (Youtube, Twitch, etc)

I only ask that you at least state the author's name, the title of the article and the URL of the article, using any style of choice.

You don’t have to include all the information in the same place if it’s not feasible. For instance, if you use the article’s imagery in a Youtube video, you may state either the author’s name or URL of the article at the bottom of the image, and then include the complete reference in the video description. In other words, for any resource used from this website, let your viewers know where it originates from.

This is a very nice example because the channel shows this website directly and their viewers know where to find it. In fact, I was so impressed with their content and commentary that I gave them an interview 🙂.

Appreciated additions

If this article has significantly contributed to your work, I would appreciate it if you could dedicate an acknowledgement section, just like I do with the people and communities that helped me.

This is of course optional and beyond the requirements of the CC license, but I think it’s a nice detail that makes us, the random authors on the net, feel part of something bigger.

Third-party publishing

If you are interested in publishing this article on a third-party website, please get in touch.

If you have translated an article and wish to publish it on a third-party website, I tend to be open about it, but please contact me first.

Sources / Keep Reading

Audio

- smspower.org, *SN76489*. ↩
- Texas Instruments, *Texas instruments SN76489*. ↩

CPU

- smspower.org, *Memory map*. ↩
- smspower.org, *Registers*. ↩
- Sega, *Sega Mk3 hardware reference manual - Rev1*. ↩
- Sega, *Service manual schematics*. ↩
- NEC, *D4168 data-sheet*. ↩

Graphics

- codeslinger.co.uk, *VDP info*. ↩
- Charles MacDonald, *Sega master system VDP documentation*. 2002. ↩
- smspower.org, *Collision detection*. ↩
- Texas Instruments, *TMS9918A, TMS9928A, TMS9929A video display processors data manual*. 1982. ↩

I/O

- smspower.org, *Pause button*. ↩

Photography

- Evan Amos, *The vanamo online game museum*. ↩
- Christopher Rivett, *Christopher rivett - computer and video game enthusiast*. ↩

Changelog

It’s always nice to keep a record of changes. For a complete report, you can check the commit log. Alternatively, here’s a simplified list:

2023-12-26

- Added a dedicated section for the Z80 CPU.

2020-11-28

- Reverted VDP data bus info after receiving more docs (see https://github.com/flipacholas/Architecture-of-consoles/issues/14), thanks

@Maxim

2020-10-17

- Corrected VDP's data bus width, thanks @ChillyWillyGuru

2020-10-14

- Minor corrections, thanks Carl Drougge

2020-10-12

- Published

2020-10-11

- Corrections and additions. Thanks @Maxim, @Charles MacDonald and @Kagesan from smspower.org; and @Mask of Destiny and @Spip from /r/Emulation discord.

2020-10-10

- First private draft finished.
- Carlos, do you know that game that has one-eyed mammoths?

« NES / Famicom Architecture PC Engine / TurboGrafx-16 Architecture »
