av-test-roms user guide

Test ROMs for emulators, for people who care about the video path rather than about emulation accuracy.

Three programs — a moving test card, a controller mapping tester, and an overscan display — built for as many consoles as have a toolchain in Homebrew. Everything is original work under MIT: no console BIOS, no commercial ROM, nothing that needs either.

They were written for cartridge, which runs a libretro core as a source inside Resolume, but there is nothing cartridge-specific in them. They are ordinary ROMs and will run in anything.

> **Before you rely on this:** all seven targets build, and **six are verified in the sense that the ROMs were run in an emulator core and the frames were looked at** — GBA in mGBA, NES in four different cores, Game Boy in gambatte, Mega Drive in two cores, Master System in Genesis Plus GX, Atari 2600 in Stella.
>
> **The C64 ROMs are built and structurally checked but have never been run**, because every C64 emulator needs a BIOS this repository will not ship. **Nothing here has been run on real console hardware** — indeed the GBA ROMs cannot be, by design; see Licensing below.
>
> This codebase was created with AI assistance, directed and reviewed by a human author.

Building

```
brew install cc65 rgbds dasm sdcc arm-none-eabi-gcc m68k-elf-gcc
make
```

`make toolchains` prints which of them you have. ROMs land in `dist/<target>/`.

Target	Toolchain	Verified in
Game Boy Advance	`arm-none-eabi-gcc`	mGBA 0.11
NES	`cc65`	fceumm, Nestopia, Mesen, QuickNES
Game Boy / GBC	`rgbds`	gambatte
Mega Drive	`m68k-elf-gcc`	PicoDrive, Genesis Plus GX
Master System	`sdcc`	Genesis Plus GX
Atari 2600	`dasm`	Stella
Commodore 64	`cc65`	built only — needs a C64 ROM set

The three programs

They are specified once, in terms of **what they must show**. The code cannot be shared between a machine that races the beam with 128 bytes of RAM and one with a display list — but the design can.

testcard

Colour bars, crosshatch, frequency burst, checkerboard and a luma/chroma edge, cycling.

What makes it worth having over a static card is that every panel carries countable motion: a marker advancing exactly one cell per emulated frame, a bar moving one pixel per frame, and a square inverting every frame.

A dropped or repeated frame is invisible in a static pattern and obvious in these. That is the whole point — count the steps, and you know whether the path is delivering every frame.

inputtest

The platform's own pad, drawn as a pad. Controls light while held and **latch once seen**, so walking every button once shows you **which one never arrived** — the actual failure when a frontend's mapping is wrong.

The raw port word is shown in hex, because **when a control lands on the wrong bit, the bit that moved tells you what it was mapped to.**

overscan

Nested safe-area insets, ruler ticks every 8 pixels from all four edges, and **asymmetric corner markers so a flipped output is obvious rather than merely plausible.**

The output is a number: *"the path is eating 6 pixels off the left"*, not *"it looks a bit off"*.

Three things the suite found while it was being written

These are the reason it exists, and they are worth reading before you use the results.

A test card can lie about its own frame rate. The GBA checkerboard first inverted with a CPU store loop — 15,360 writes a frame — and sustained about a third of frame rate, so the ticker repeated cells instead of advancing one per frame. **An instrument whose frame counter is wrong is worse than no instrument.** It is DMA now, and holds 39/39 single-cell steps on every panel.

Cores disagree about how much of the NES picture there is. fceumm, Nestopia and Mesen all return 256×240; **QuickNES returns 240×224**, cropping eight pixels off each edge and swallowing the title row. Three agree and one does not — and `overscan` puts a number on the difference. That is precisely the question this suite exists to answer.

One Game Boy binary can serve both machines. Bar *i* uses the tile whose ink is index *i* mod 4 and colour palette *i*, so a Game Boy Color shows eight colours and a DMG — where the attribute map does not exist — falls back to its four shades twice over, with adjacent bars still distinguishable. No second code path, nothing drawn twice.

Licensing, and what is deliberately missing

MIT, and every byte is original. Three consequences worth stating plainly:

- **The GBA ROMs will not boot on real hardware.** A GBA BIOS checks the Nintendo logo bitmap in the cartridge header before it runs anything. That bitmap is Nintendo's artwork, so the field is zeroed. **They run in any emulator that skips the check.**
- **mGBA will not load the Game Boy ROMs**, for the same reason: its GB core identifies a ROM *by* that logo. **gambatte and SameBoy load them fine**, and `rgbfix -f l` on your own copy makes mGBA accept it — your call, on your machine, not something this repo ships.
- **No cores are included**, and none will be. Cores have their own licences, several of them non-commercial.

If something looks wrong

Symptom	Cause
The ticker repeats cells	The path is dropping or repeating frames — which is what the countable motion is for.
A GBA ROM will not boot on hardware	By design. The logo field is zeroed.
mGBA refuses a Game Boy ROM	Same reason. Use gambatte or SameBoy.
The NES picture is 240×224	QuickNES crops eight pixels off each edge. Three other cores return 256×240.
A pad button never lights	That mapping never arrived. The latch is what makes this visible after one pass.
The C64 ROMs will not run	They have never been run here either — a C64 emulator needs a BIOS this repo will not ship.