ChatGPT

Extracting SNES Sprite Data and ROM Offsets with Mesen 2 Lua

Mesen 2 Lua API for SNES Memory Access

Mesen 2 provides a rich Lua API for accessing various SNES memory regions, which is crucial for mapping sprite data to ROM offsets. In the SNES context, relevant memory types (`emu.memType`) include:

- `snesVideoRam` – SNES VRAM (Video RAM, 64KB) where tile graphics are stored [1].
- `snesSpriteRam` – SNES OAM (Object Attribute Memory) for sprite attributes (544 bytes for 128 sprites) [1].
- `snesCgRam` – SNES CGRAM (Color Palette RAM, 512 bytes) for palettes [1].
- `snesPrgRom` – The entire SNES ROM data (PRG ROM) loaded by the emulator [2].

Using the Lua API, you can **read from** these memory regions with functions like `emu.read(address, memoryType)` for bytes (and `read16`, `read32` for larger values) [3]. For example, to read a byte from VRAM at a given address:

```
byte = emu.read(vramAddress, emu.memType.snesVideoRam)
```

Note: When reading SNES CPU memory-mapped registers (like PPU registers), use the “debug” memory types to avoid side effects [4]. For instance, reading from `emu.memType.snesDebug` allows you to peek the last written values of write-only registers safely. This will be useful for fetching PPU register values such as sprite base addresses without disturbing the emulator state.

Reading Sprite OAM Data and PPU Registers via Lua

Each SNES sprite (OBJ) has a 4-byte entry in OAM containing its position and tile info. We can use Lua to retrieve these entries and the relevant PPU settings:

1. **Access OAM (Sprite RAM):** Loop through the OAM entries using `emu.read` on `snesSpriteRam`. Each sprite record consists of: Y position, X position, tile index, and attribute byte [5]. The *attribute* byte encodes flags: horizontal/vertical flip, priority, the **palette number** (`ppp` bits), and the **name table bit** (`N`) [6]. For example:

```
-- Read all 128 sprite entries (4 bytes each)
local oam = {}
for i = 0, 127 do
    local base = i * 4
    oam[i] = {
        Y = emu.read(base + 0, emu.memType.snesSpriteRam),
        X = emu.read(base + 1, emu.memType.snesSpriteRam),
```

```
-- low 8 bits of X
        Tile = emu.read(base + 2, emu.memType.snesSpriteRam),
-- tile index (low 8 bits)
        Attr = emu.read(base + 3, emu.memType.snesSpriteRam)
-- attributes (flip, priority, palette, N)
    }
}
```

The **palette number** can be extracted as `paletteNum = (Attr >> 1) & 0x07` (bits 1–3) and the **N flag** as `N = Attr & 0x01` [6]. According to SNES specs, sprite palettes are stored in CGRAM starting at index 128, and `ppp` gives which 16-color palette (0–7) the sprite uses (colors 128 + 16*ppp to 128 + 16*ppp + 15) [6] [7].

2. **Get PPU Sprite Table Settings:** Read the SNES PPU register **$2101 (OBSEL)** in debug mode to determine where sprite graphics are loaded in VRAM. This register contains the **Name Base** and **Name Select** fields for sprites [8] [9]. In $2101:

3. Bits `bbb` (bits 0–2) = Name Base select (the base VRAM address for sprite patterns).
4. Bits `nn` (bits 3–4) = Name Select (the size/offset of the second sprite graphics table).
   We can retrieve this with:

```
local obsel = emu.read(0x2101, emu.memType.snesDebug)
local baseBits = obsel & 0x07        -- bbb: sprite VRAM base pointer
(in 8 KB units)
local nameSel  = (obsel >> 3) & 0x03 -- nn: index for second name table
offset
```

These values will be used to calculate the actual VRAM address of a sprite's tile data.

Calculating VRAM Addresses for Sprite Tiles

Using the sprite's OAM data and the PPU settings, we can compute the VRAM address where the sprite's graphics are stored. The SNES divides sprite graphics into up to two character tables in VRAM. The **Name Base** from $2101 gives the starting address of the first table, and the **Name Select** together with the sprite's `N` bit determines the offset if the sprite uses the second table [10]. According to the official formula:

VRAM word address of a sprite's first tile = `((Base << 13) + (tileIndex << 4) + (N ? ((Name+1) << 12) : 0)) & 0x7FFF` [10]

In this formula, `Base` is the baseBits from $2101, `Name` is nameSel, `tileIndex` is the 8-bit tile number from OAM, and `N` is the sprite's name-table flag. This calculation yields a **word address** (16-bit words) in VRAM. You can convert it to a byte address by multiplying by 2. For example:

```
-- Compute VRAM byte address for the sprite's tile
local baseAddrWords = baseBits << 13         -- Base * 8192 (word address)
local tileAddrWords = sprite.Tile << 4
-- tileIndex * 16 (word address, 16 words per 8x8 tile)
```

```
if N ~= 0 then
    -- If sprite uses second character table, add offset (NameSel+1)*4096
words
    tileAddrWords = tileAddrWords + ((nameSel + 1) << 12)
end
local vramAddrWords = (baseAddrWords + tileAddrWords) & 0x7FFF
local vramAddrBytes = vramAddrWords * 2       -- convert word address to byte
address
```

This gives the starting VRAM address of the sprite's tile graphics. Keep in mind sprite size: if the sprite is larger than 8x8 (OAM size bit set and $2101's size field `sss`), it will consume multiple tiles in VRAM (e.g. a 16×16 sprite uses 4 tiles). In such cases, the OAM tile index refers to the first tile, and the following tiles are stored contiguously (with wrapping inside the 16×16 tile table) [11] [12]. You may want to account for sprite dimensions and read a larger block of VRAM if you need the full graphics of a big sprite. For a simple case, reading one 8×8 tile (32 bytes for 4bpp) from `vramAddrBytes` is enough to find the base graphics data. For example:

```
-- Read 32 bytes (one 8x8 tile) from VRAM
local tileData = {}
for i = 0, 31 do
    tileData[i] = emu.read(vramAddrBytes + i, emu.memType.snesVideoRam)
end
```

Mapping Sprite Graphics to ROM Offsets

Once you have the sprite's graphical data from VRAM, the next step is to locate that data in the ROM. If the game stores sprite graphics uncompressed, the byte sequence for the tile(s) will appear in the ROM's data. You can scan the ROM (`snesPrgRom`) for the 32-byte (or larger) pattern to find its **file offset**. For example:

```
-- Naive search for tileData bytes in ROM
local romSize =  emu.getRomInfo().size  -- (if size is provided; if not, use
known length or loop until reads fail)
local foundOffset = nil
for addr = 0, romSize-#tileData do
    local match = true
    for i,byte in ipairs(tileData) do
        if emu.read(addr + i - 1, emu.memType.snesPrgRom) ~= byte then
            match = false
            break
        end
    end
    if match then foundOffset = addr; break end
end
if foundOffset then
    print(string.format("Sprite tile 0x%02X found at ROM offset $%06X",
```

```
sprite.Tile, foundOffset))
end
```

This will brute-force search the ROM for the sprite's graphics. In practice, you might want to optimize this (e.g. search for a smaller unique subset, or use Lua string functions if the API allows). **Note:** This approach only works if the sprite graphics are stored plainly in the ROM. Many games (especially those using compression or special chips) may not have 1:1 VRAM-to-ROM byte patterns [13]. In Kirby Super Star (a SA-1 game), graphics could be compressed; if so, you'd need a different strategy (e.g. tracing the decompression routine or using Mesen's debugging features to catch DMA transfers from ROM to VRAM).

Using Access Counters (Alternative Method)

Mesen 2 also offers **access counters** that can help identify where sprite data came from in ROM without manual searching. By resetting counters and then checking which ROM addresses were read during the frame a sprite loaded, you can pinpoint the source. For example, you can use:

```
emu.resetAccessCounters()
-- (Run one frame or step until sprite appears)
local lastRead = emu.getAccessCounters(emu.counterType.lastReadClock,
emu.memType.snesPrgRom)
```

This returns an array of timestamps for each byte in the PRG ROM, indicating when each was last read [14] [15]. The highest (most recent) values in that array correspond to ROM locations the game just accessed. By correlating these with VRAM writes, you can deduce which ROM bytes were DMA'd or copied to VRAM for the sprite. This is an advanced technique, but it can be very powerful for real-time analysis (e.g. finding graphics or palette data loaded via DMA).

Extracting Palette Data and ROM Offset

You can apply a similar approach for sprite **palette** data: each sprite's palette number (`ppp`) tells you which 16-color palette in CGRAM it uses [6] [7]. You can read the 16 colors from CGRAM via `emu.read` on `snesCgRam`. For example, palette 5 (ppp=5) starts at CGRAM index 128+5*16 = 208, so its 16 colors occupy bytes $D0–$EF in CGRAM (since each color is 2 bytes). You could read those 32 bytes and search the ROM for that sequence as well. Many games store palettes in the ROM as 16-word tables, so if they are not dynamically generated, the exact 32-byte sequence of the palette might appear in the ROM. Searching the ROM for the palette bytes can reveal the palette's offset in the file. If the palette is loaded via DMA, using the access counter method (checking which ROM addresses were read into CGRAM) is another reliable strategy.

Automating the Process in Real-Time

To continuously extract sprite info as the game runs (e.g. log Kirby or enemy sprites and their ROM locations in real time), you can leverage **frame callbacks** in Mesen. The function `emu.addEventCallback(callbackFunction, eventType)` allows you to run code at specified emulator events [16]. For instance, to run code at the end of every frame (once per frame during vertical blank), you can do:

```lua
emu.addEventCallback(function(cpu)
    -- (Your code to read OAM, compute VRAM addresses, search ROM, etc.)
    -- For example, identify a particular sprite (by ID or position) and
print its ROM offset:
    local kirbyIndex = 0  -- suppose sprite 0 is Kirby
    local tile = emu.read(kirbyIndex*4 + 2, emu.memType.snesSpriteRam)
    local attr = emu.read(kirbyIndex*4 + 3, emu.memType.snesSpriteRam)
    local N    = attr & 0x01
    local pal  = (attr >> 1) & 0x07
    -- ...compute VRAM address as shown above...
    -- ...search ROM for tile bytes...
    print(string.format("Sprite#%d (tile $%02X, palette %d) -> ROM $%06X",
kirbyIndex, tile, pal, foundOffset or 0))
end, emu.eventType.endFrame)
```

Here we use the `endFrame` event, which triggers after each frame is rendered [17]. You could also use `eventType.startFrame` or even an NMI callback, but `endFrame` ensures that the sprite and palette for the current frame are fully loaded (since sprites are usually transferred to VRAM during VBlank). The callback can iterate through OAM to log any sprite of interest (Kirby, enemies, etc.) and output their ROM offsets.

By combining these Lua API features, you can dynamically **convert sprite IDs/tile data to their ROM offsets**. In summary, you will:

- Read sprite OAM entries in Lua (get tile index and palette info).
- Read PPU register $2101 (sprite VRAM layout) via debug memory to get base addresses.
- Compute the VRAM address of the sprite's graphics using the formula with OAM tile and N flag [10].
- Read the sprite's graphic bytes from VRAM and locate that sequence in the ROM (`snesPrgRom`) to get the file offset.
- Do the same for the palette data from CGRAM if needed.
- (Optionally, use Mesen's event callbacks to automate this each frame, and use access counters or memory callbacks to streamline finding the ROM source.)

With this approach, for example, you can find that a sprite (like Kirby or an enemy) displayed on screen corresponds to a specific location in the Kirby Super Star ROM. Each sprite's tile graphics and palette can be traced back to their ROM offsets using Mesen 2's Lua scripting facilities, allowing for efficient hacking or analysis of the game's assets.

Sources: Mesen 2 Lua API documentation [1] [4]; SNES technical manuals for OAM/VRAM usage [6] [10]; community insights on graphics extraction [13].

---

1 2 3 4 14 15 16 17 Mesen Lua API reference_Mesen Lua 函数库 - Pastebin.com
https://pastebin.com/z9w7Dj5Q

5 6 7 10 11 12 Sprites | Super Famicom Development Wiki
https://wiki.superfamicom.org/sprites

8 9 PPU Registers - SnesLab
https://sneslab.net/wiki/PPU_Registers

13 Mesen-S - SNES Emulator - Page 21 - nesdev.org
https://forums.nesdev.org/viewtopic.php?t=18658&start=300