

SNES Registers

This page is based on the great work of [Anomie](#), [Qwertie](#) and [Martin Korth](#). If this document helps you and you feel like giving back, consider a donation to the [Nocash Project](#), because a lot of info on this page has been taken from there.

Address Bus B Registers

| Register | Address | Name | Style | Access | Timing |
|--|------------------------|--------------------|--------|--------|---------------------------|
| Screen Display | \$2100 | INIDISP | single | write | any time |
| Object Size and Object | \$2101 | OBSEL | single | write | f-blank, v-blank |
| OAM Address and Priority Rotation (Low) | \$2102 | OAMADDL | single | write | f-blank, v-blank |
| OAM Address and Priority Rotation (High) | \$2103 | OAMADDH | single | write | f-blank, v-blank |
| OAM Data Write | \$2104 | OAMDATA | single | write | f-blank, v-blank |
| BG Mode and BG Character Size | \$2105 | BGMODE | single | write | f-blank, v-blank, h-blank |
| Mosaic Size and Mosaic Enable | \$2106 | MOSAIC | single | write | f-blank, v-blank, h-blank |
| BG1 Screen Base and Screen Size | \$2107 | BG1SC | single | write | f-blank, v-blank |
| BG2 Screen Base and Screen Size | \$2108 | BG2SC | single | write | f-blank, v-blank |
| BG3 Screen Base and Screen Size | \$2109 | BG3SC | single | write | f-blank, v-blank |
| BG4 Screen Base and Screen Size | \$210A | BG3SC | single | write | f-blank, v-blank |
| BG Character Data Area Designation (BG1 & BG2) | \$210B | BG12NBA | single | write | f-blank, v-blank |
| BG Character Data Area Designation (BG3 & BG4) | \$210C | BG34NBA | single | write | f-blank, v-blank |
| BG1 and Mode 7 Horizontal Scroll | \$210D | BG1HOFS and M7HOFS | dual | write | f-blank, v-blank, h-blank |
| BG1 and Mode 7 Vertical Scroll | \$210E | BG1VOFS and M7VOFS | dual | write | f-blank, v-blank, h-blank |
| BG2 Horizontal Scroll | \$210F | BG2HOFS | dual | write | f-blank, v-blank, h-blank |
| BG2 Vertical Scroll | \$2110 | BG2VOFS | dual | write | f-blank, v-blank, h-blank |
| BG3 Horizontal Scroll | \$2111 | BG3HOFS | dual | write | f-blank, v-blank, h-blank |
| BG3 Vertical Scroll | \$2112 | BG3VOFS | dual | write | f-blank, v-blank, h-blank |
| BG4 Horizontal Scroll | \$2113 | BG4HOFS | dual | write | f-blank, v-blank, h-blank |

| Register | Address | Name | Style | Access | Timing |
|-------------------------------------|------------------------|---------|--------|--------|---------------------------|
| BG4 Vertical Scroll | \$2114 | BG4VOFS | dual | write | f-blank, v-blank, h-blank |
| Video Port Control | \$2115 | VMAIN | single | write | f-blank, v-blank |
| VRAM Address (Low) | \$2116 | VMADDL | single | write | f-blank, v-blank |
| VRAM Address (High) | \$2117 | VMADDH | single | write | f-blank, v-blank |
| VRAM Data Write (Low) | \$2118 | VMDATAL | single | write | f-blank, v-blank |
| VRAM Data Write (High) | \$2119 | VMDATAH | single | write | f-blank, v-blank |
| Mode 7 Settings | \$211A | M7SEL | single | write | f-blank, v-blank |
| Mode 7 Matrix A | \$211B | M7A | dual | write | f-blank, v-blank, h-blank |
| Mode 7 Matrix B | \$211C | M7B | dual | write | f-blank, v-blank, h-blank |
| Mode 7 Matrix C | \$211D | M7C | dual | write | f-blank, v-blank, h-blank |
| Mode 7 Matrix D | \$211E | M7D | dual | write | f-blank, v-blank, h-blank |
| Mode 7 Center X | \$211F | M7X | dual | write | f-blank, v-blank, h-blank |
| Mode 7 Center Y | \$2120 | M7Y | dual | write | f-blank, v-blank, h-blank |
| CGRAM Address | \$2121 | CGADD | single | write | f-blank, v-blank, h-blank |
| CGRAM Data Write | \$2122 | CGDATA | dual | write | f-blank, v-blank, h-blank |
| Window Mask Settings (BG1 & BG2) | \$2123 | W12SEL | single | write | f-blank, v-blank, h-blank |
| Window Mask Settings (BG3 & BG4) | \$2124 | W34SEL | single | write | f-blank, v-blank, h-blank |
| Window Mask Settings (OBJ and MATH) | \$2125 | WOBJSEL | single | write | f-blank, v-blank, h-blank |
| Window 1 Left Position | \$2126 | WH0 | single | write | f-blank, v-blank, h-blank |
| Window 1 Right Position | \$2127 | WH1 | single | write | f-blank, v-blank, h-blank |
| Window 2 Left Position | \$2128 | WH2 | single | write | f-blank, v-blank, h-blank |
| Window 2 Right Position | \$2129 | WH3 | single | write | f-blank, v-blank, h-blank |
| Window Mask Logic (BGs) | \$212A | WBGLOG | single | write | f-blank, v-blank, h-blank |
| Window Mask Logic (OBJ and MATH) | \$212B | WOBJLOG | single | write | f-blank, v-blank, h-blank |
| Screen Destination Registers | \$212C | TM | single | write | f-blank, v-blank, h-blank |
| Screen Destination Registers | \$212D | TS | single | write | f-blank, v-blank, h-blank |
| Window Mask Destination Registers | \$212E | TMW | single | write | f-blank, v-blank, h-blank |

| Register | Address | Name | Style | Access | Timing |
|--|---------|-------------|--------|--------|---------------------------|
| Window Mask Destination Registers | \$212F | TSW | single | write | f-blank, v-blank, h-blank |
| Color Math Registers | \$2130 | CGWSEL | single | write | f-blank, v-blank, h-blank |
| Color Math Registers | \$2131 | CGADSUB | single | write | f-blank, v-blank, h-blank |
| Color Math Registers | \$2132 | COLDATA | single | write | f-blank, v-blank, h-blank |
| Screen Mode Select Register | \$2133 | SETINI | single | write | f-blank, v-blank, h-blank |
| Multiplication Result Registers | \$2134 | MPYL | single | read | f-blank, v-blank, h-blank |
| Multiplication Result Registers | \$2135 | MPYM | single | read | f-blank, v-blank, h-blank |
| Multiplication Result Registers | \$2136 | MPYH | single | read | f-blank, v-blank, h-blank |
| Software Latch Register | \$2137 | SLHV | single | | any time |
| OAM Data Read Register | \$2138 | OAMDATAREAD | dual | read | f-blank, v-blank |
| VRAM Data Read Register (Low) | \$2139 | VMDATALREAD | single | read | f-blank, v-blank |
| VRAM Data Read Register (High) | \$213A | VMDATAHREAD | single | read | f-blank, v-blank |
| CGRAM Data Read Register | \$213B | CGDATAREAD | dual | read | f-blank, v-blank |
| Scanline Location Registers (Horizontal) | \$213C | OPHCT | dual | read | any time |
| Scanline Location Registers (Vertical) | \$213D | OPVCT | dual | read | any time |
| PPU Status Register | \$213E | STAT77 | single | read | any time |
| PPU Status Register | \$213F | STAT78 | single | read | any time |
| APU IO Registers | \$2140 | APUIO0 | single | both | any time |
| APU IO Registers | \$2141 | APUIO1 | single | both | any time |
| APU IO Registers | \$2142 | APUIO2 | single | both | any time |
| APU IO Registers | \$2143 | APUIO3 | single | both | any time |
| WRAM Data Register | \$2180 | WMDATA | single | both | any time |
| WRAM Address Registers | \$2181 | WMADDL | single | write | any time |
| WRAM Address Registers | \$2182 | WMADDM | single | write | any time |
| WRAM Address Registers | \$2183 | WMADDH | single | write | any time |

Old Style Joypad Registers

| Register | Address | Name | Style | Access | Timing |
|----------------------------|---------|---------|----------------|------------|----------------------------------|
| Old Style Joypad Registers | \$4016 | JOYSER0 | single (write) | read/write | any time that is not auto-joypad |
| Old Style Joypad Registers | \$4017 | JOYSER1 | many (read) | read | any time that is not auto-joypad |

Internal CPU Registers

| Register | Address | Name | Style | Access | Timing |
|--|---------|----------|--------|--------|----------------------------------|
| Interrupt Enable Register | \$4200 | NMITIMEN | single | write | any time |
| IO Port Write Register | \$4201 | WRIO | single | write | any time |
| Multiplicand Registers | \$4202 | WRMPYA | single | write | any time |
| Multiplicand Registers | \$4203 | WRMPYB | single | write | any time |
| Divisor & Dividend Registers | \$4204 | WRDIVL | single | write | any time |
| Divisor & Dividend Registers | \$4205 | WRDIVH | single | write | any time |
| Divisor & Dividend Registers | \$4206 | WRDIVB | single | write | any time |
| IRQ Timer Registers (Horizontal - Low) | \$4207 | HTIMEL | single | write | any time |
| IRQ Timer Registers (Horizontal - High) | \$4208 | HTIMEH | single | write | any time |
| IRQ Timer Registers (Vertical - Low) | \$4209 | VTIMEL | single | write | any time |
| IRQ Timer Registers (Vertical - High) | \$420A | VTIMEH | single | write | any time |
| DMA Enable Register | \$420B | MDMAEN | single | write | any time |
| HDMA Enable Register | \$420C | HDMAEN | single | write | any time |
| ROM Speed Register | \$420D | MEMSEL | single | write | any time |
| Interrupt Flag Registers | \$4210 | RDNMI | single | read | any time |
| Interrupt Flag Registers | \$4211 | TIMEUP | single | read | any time |
| PPU Status Register | \$4212 | HVBJOY | single | read | any time |
| IO Port Read Register | \$4213 | RDIO | single | read | any time |
| Multiplication Or Divide Result Registers (Low) | \$4214 | RDDIVL | single | read | any time |
| Multiplication Or Divide Result Registers (High) | \$4215 | RDDIVH | single | read | any time |
| Multiplication Or Divide Result Registers (Low) | \$4216 | RDMPYL | single | read | any time |
| Multiplication Or Divide Result Registers (High) | \$4217 | RDMPYH | single | read | any time |
| Controller Port Data Registers (Pad 1 - Low) | \$4218 | JOY1L | single | read | any time that is not auto-joypad |
| Controller Port Data Registers (Pad 1 - High) | \$4219 | JOY1H | single | read | any time that is not auto-joypad |
| Controller Port Data Registers (Pad 2 - Low) | \$421A | JOY2L | single | read | any time that is not auto-joypad |
| Controller Port Data Registers (Pad 2 - High) | \$421B | JOY2H | single | read | any time that is not auto-joypad |
| Controller Port Data Registers (Pad 3 - Low) | \$421C | JOY3L | single | read | any time that is not auto-joypad |
| Controller Port Data Registers (Pad 3 - High) | \$421D | JOY3H | single | read | any time that is not auto-joypad |
| Controller Port Data Registers (Pad 4 - Low) | \$421E | JOY4L | single | read | any time that is not auto-joypad |
| Controller Port Data Registers (Pad 4 - High) | \$421F | JOY4H | single | read | any time that is not auto-joypad |

DMA Registers

| Register | Address | Name |
|------------------------------|---------|------|
| DMA Control Register | \$43×0 | DMAP |
| DMA Destination Register | \$43×1 | BBAD |
| DMA Source Address Registers | \$43×2 | A1Tx |
| DMA Source Address Registers | \$43×3 | A1Tx |
| DMA Source Address Registers | \$43×4 | A1B |
| DMA Size Registers (Low) | \$43×5 | DASx |
| DMA Size Registers (High) | \$43×6 | DASx |

HDMA Registers

| Register | Address | Name |
|---|---------|------|
| HDMA Control Register | \$43×0 | DMAP |
| HDMA Destination Register | \$43×1 | BBAD |
| HDMA Table Address Registers | \$43×2 | A1Tx |
| HDMA Table Address Registers | \$43×3 | A1Tx |
| HDMA Table Address Registers | \$43×4 | A1B |
| HDMA Indirect Address Registers | \$43×5 | DASx |
| HDMA Indirect Address Registers | \$43×6 | DASx |
| HDMA Indirect Address Registers | \$43×7 | DASB |
| HDMA Mid Frame Table Address Registers (Low) | \$43×8 | A2Ax |
| HDMA Mid Frame Table Address Registers (High) | \$43×9 | A2Ax |
| HDMA Line Counter Register | \$43xA | NTLR |

Register Details

Format:

| | |
|---------|------|
| rw?fvha | Name |
| | bits |

“Name” is the official and unofficial name of the register.
“bits” is either 8 or 16 characters explicating the bitfields in this register.

The flags are:

| | |
|---------|--|
| rw?fvha | |
| +--> | '+' if it can be read/written at any time, '-' otherwise |
| +---> | '+' if it can be read/written during H-Blank |
| +----> | '+' if it can be read/written during V-Blank |
| +-----> | '+' if it can be read/written during force-blank |
| +-----> | Read/Write style: 'b' => byte |
| | 'h'/'l' => read/write high/low byte of a word |
| | 'w' => word read/write twice low then high |

```
|+-----> 'w' if the register is writable for an effect
+-----> 'r' if the register is readable for a value or effect (i.e. not
         open bus).
```

Screen Display

```
$2100 wb++++ INIDISP
      xuuubbbb

      x    = Forced Blanking (0=Normal, 1=Screen Black)
      uuu  = Unused
      bbbb = Master Brightness (0=Screen Black, or N=1..15:
      Brightness*(N+1)/16)
```

This register is used for screen fades. In Forced Blank, VRAM, OAM and CGRAM can be freely accessed (otherwise it's accessible only during Vblank). Even when in forced blank, the TV Set keeps receiving Vsync/Hsync signals (thus producing a stable black picture). And, the CPU keeps receiving Hblank/Vblank signals (so any enabled video NMIs, IRQs, HDMA's are kept generated).

Note that force blank CAN be disabled mid-scanline. However, this can result in glitched graphics on that scanline, as the internal rendering buffers will not have been updated during force blank. Current theory is that BGs will be glitched for a few tiles (depending on how far in advance the PPU operates), and OBJ will be glitched for the entire scanline.

Also, writing this register on the first line of V-Blank (225 or 240, depending on overscan) when force blank is currently active causes the OAM Address Reset to occur.

[Back to top](#)

Object Size and Object Base

```
$2101 wb++?- OBSEL
      sssnnbbb
      sss = OBJ Size Selection (0-5, see below) (6-7=Reserved)
      Val Small Large
      000 = 8x8   16x16   ;Caution:
      001 = 8x8   32x32   ;In 224-lines mode, OBJs with 64-pixel
height
      010 = 8x8   64x64   ;may wrap from lower to upper screen
border.
      011 = 16x16 32x32   ;In 239-lines mode, the same problem
applies
      100 = 16x16 64x64   ;also for OBJs with 32-pixel height.
      101 = 32x32 64x64
      110 = 16x32 32x64 (undocumented)
      111 = 16x32 32x32 (undocumented)
      (Ie. a setting of 0 means Small OBJs=8x8, Large OBJs=16x16)
```

```

pixels)
    (Whether an OBJ is "small" or "large" is selected by a bit in
OAM)
    nn = Gap between OBJ 0FFh and 100h (0=None) (4K-word steps) (8K-
byte steps)
    bbb = Base Address for OBJ Tiles 000h..0FFh (8K-word steps) (16K-
byte steps) (Addr>>14)

```

This register selects the location in VRAM where the character data is stored, and the size of sprites on the screen. The byte location of the character data can be found by shifting the b (base selection) bits left by 14. Note that this allows only four different locations in VRAM to put the sprite data; the high bit of the base selection should always be zero since only 64K of VRAM can be addressed.

[Back to top](#)

OAM Address and Priority Rotation

```

$2102  wl++?- OAMADDL
$2103  wh++?- OAMADDH
        puuuuuub aaaaaaaaa
        p          = OAM Priority Rotation (0=OBJ #0, 1=OBJ #N) (OBJ with
highest priority)
        uuuuuu     = Unused
        baaaaaaaa  = OAM Address (for OAM read/write)
        aaaaaaa    = OBJ Number #N (for OBJ Priority) (bit 7-1 are used for
two purposes)

```

This register contains of a 9 bit Reload value and a 10 bit Address register (plus the priority flag). Writing to \$2102 or \$2103 does change the lower 8 bit or upper 1 bit of the Reload value, and does additionally copy the (whole) 9 bit Reload value to the 10 bit Address register (with address Bit 0=0 so next access will be an even address). When OAM Priority Rotation bit is set, an Obj other than Sprite 0 may be given priority.

OAM address can be thought of in two ways, depending on your conception of OAM. If you consider OAM as a 544-byte table, baaaaaaaa is the word address into that table. If you consider OAM to be a 512-byte table and a 32-byte table, b is the table selector and aaaaaaaa is the word address in the table.

During rendering, the PPU is destroying the Address register (using it internally for whatever purposes), after rendering (at begin of Vblank, ie. at begin of line 225/240, but only if not in Forced Blank mode) it reinitializes the Address from the Reload value; the same reload occurs also when deactivating forced blank anytime during the first scanline of vblank (ie. during line 225/240). This is known as 'OAM reset'. 'OAM reset' also occurs on certain writes to \$2100.

Writing to either \$2102 or \$2103 resets the entire internal OAM Address to the values last written to this register. E.g., if you set \$0104 to this register, write 4 bytes, then write \$01 to \$2103, the internal OAM address will point to word 4, not word 6.

[Back to top](#)

OAM Data Write

```
$2104 wb+-- OAMDATA
      dddddddd = byte to write to VRAM
      Writes to EVEN and ODD byte-addresses work as follows:
          Write to EVEN address    -->  set OAM_Lsb = Data      ;memorize
value
          Write to ODD address<200h -->  set WORD[addr-1] = Data*256 +
OAM_Lsb
          Write to ANY address>1FFh -->  set BYTE[addr] = Data
```

This register writes a byte to OAM. After the byte is stored, the OAM address is incremented so that the next write or read will be to the following address. Note that OAM writes are done in an odd manner, in particular the low table of OAM is not affected until the high byte of a word is written (however, the high table is affected immediately). Thus, if you set the address, then alternate writes and reads, OAM will never be affected until you reach the high table!

Similarly, if you set the address to 0, then write 1, 2, read, then write 3, OAM will end up as "01 02 01 03", rather than "01 02 xx 03" as you might expect.

Technically, this register CAN be written during H-blank (and probably mid-scanline as well). However, due to OAM address invalidation the actual OAM byte written will probably not be what you expect. Note that writing during force-blank will only work as expected if that force-blank was begun during V-Blank, or (probably) if \$2102-\$2103 have been reset during that force-blank period. OAM Size is \$0220 bytes (addresses \$0220..\$03FF are mirrors of \$0200..\$021F)

[Back to top](#)

BG Mode and BG Character Size

```
$2105 wb+++ BGMODE
      DCBAemmm
      D  = BG tile size for BG4 (0=8x8, 1=16x16) (BgMode0..4: variable
8x8 or 16x16)
      C  = BG tile size for BG3 (0=8x8, 1=16x16) (BgMode5: 8x8 acts as
16x8)
      B  = BG tile size for BG2 (0=8x8, 1=16x16) (BgMode6: fixed 16x8?)
      A  = BG tile size for BG1 (0=8x8, 1=16x16) (BgMode7: fixed 8x8)
      e  = Mode 1 BG3 priority bit (0=Normal, 1=High)
      mmm = BG Mode (0..7, see below)

      Mode      BG depth  OPT  Priorities
                1 2 3 4      Front -> Back Type
      -----
=====
      0          2 2 2 2      n      3AB2ab1CD0cd Normal
      1          4 4 2        n      3AB2ab1C 0c Normal
                * if e set: C3AB2ab1 0c Normal
      2          4 4          y      3A 2B 1a 0b Offset-per-tile
```


| | | | | | |
|--|---------|-----|---|-------------|---------------------------|
| | 3 | 8 4 | n | 3A 2B 1a 0b | Normal |
| | 4 | 8 2 | y | 3A 2B 1a 0b | Offset-per-tile |
| | 5 | 4 2 | n | 3A 2B 1a 0b | 512-pix-hires |
| tile | 6 | 4 | y | 3A 2 1a 0 | 512-pix- plus Offset-per- |
| | 7 | 8 | n | 3 2 1a 0 | Rotation /Scaling |
| | 7+EXTBG | 8 7 | n | 3 2B 1a 0b | Rotation /Scaling |
| Mode 5/6 don't support screen addition/subtraction. | | | | | |
| CG Direct Select is support on BG1 of Mode 3/4, and on BG1/BG2? of Mode 7. | | | | | |

This register determines the size of tile represented by one entry in the tile map array, the order that BGs are drawn on the screen, and the screen mode. If the BG tile size for BG1/BG2/BG3/BG4 bit is set, then the BG is made of 16×16 tiles. Otherwise, 8×8 tiles are used. However, note that Modes 5 and 6 always use 16-pixel wide tiles, and Mode 7 always uses 8×8 tiles. “OPT” means “Offset-per-tile mode”. For the priorities, numbers mean sprites with that priority. Letters correspond to BGs (A=1, B=2, etc), with upper/lower case indicating tile priority 1/0. The priority bit only works in Mode 1. In all other modes, it is ignored (drawing is performed as if this bit were clear.)

Notice that Mode 7 has only one BG. All games which appear to have a Mode 7 screen but more than one BG either use sprites to simulate a BG, or switch video modes midframe via HDMA. Mode 7's EXTBG mode allows you to enable BG2, which uses the same tilemap and character data as BG1 but interprets bit 7 of the pixel data as a priority bit.

[Back to top](#)

Mosaic Size and Mosaic Enable

\$2106 wb+++ - MOSAIC

xxxxDCBA

xxxx = Mosaic Size (0=Smallest/1x1, 0Fh=Largest/16x16)

D = BG4 Mosaic Enable (0=0ff, 1=0n)

C = BG3 Mosaic Enable (0=0ff, 1=0n)

B = BG2 Mosaic Enable (0=0ff, 1=0n)

A = BG1 Mosaic Enable (0=0ff, 1=0n)

Allows to divide the BG layer into NxN pixel blocks, in each block, the hardware picks the upper-left pixel of each block, and fills the whole block by the color - thus effectively reducing the screen resolution.

Horizontally, the first block is always located on the left edge of the TV screen. Vertically, the first block is located on the top of the TV screen. When changing the mosaic size mid-frame, the hardware does first finish current block (using the old vertical size) before applying the new vertical size. Technically, vertical mosaic is implemented as so: subtract the vertical index (within the current block) from the vertical scroll register (BGnVOFS).

It seems that writing the same value to this register does not reset the 'starting scanline'. Note that mosaic is applied after scrolling, but before any clip windows, color windows, or math. So the XxX block can be partially clipped, and it can be mathed as normal with a non-mosaiced BG. But scrolling can't make it partially one color and partially another.

Modes 5-6 should 'double' the expansion factor to expand half-pixels. This actually makes xxxx=0 have a visible effect, since the even half-pixels (usually on the subscreen) hide the odd half-pixels. The same thing happens vertically with interlace mode.

Mode 7, of course, is weird. BG1 mosaics about like normal, as long as you remember that the Mode 7 transformations have no effect on the XxX blocks. BG2 uses bit A to control 'vertical mosaic' and bit B to control 'horizontal mosaic', so you could be expanding over 1xX, Xx1, or XxX blocks. This can get really interesting as BG1 still uses bit A as normal, so you could have the BG1 pixels expanded XxX with high-priority BG2 pixels expanded 1xX on top of them.

[Back to top](#)

BG Screen Base and Screen Size

```
$2107  wb++?- BG1SC
$2108  wb++?- BG2SC
$2109  wb++?- BG3SC
$210A  wb++?- BG4SC
      aaaaaabb
      aaaaaa = Screen Base Address in VRAM (in 1K-word steps, aka 2K-byte
steps) (Addr>>10)
      bb      = Screen Size: 00=32x32 tiles (One-Screen)
                                01=64x32 tiles (V-Mirror)
                                10=32x64 tiles (H-Mirror)
                                11=64x64 tiles (Four-Screen)
                                (00:SC0 SC0   01:SC0 SC1  10:SC0 SC0  11:SC0
SC1)
                                (  SC0 SC0      SC0 SC1      SC1 SC1      SC2
SC3)
```

Specifies the BG Map addresses in VRAM. The "SCn" screens consists of 32x32 tiles each. Ignored in Mode 7 (Base is always zero, size is always 128x128 tiles).

To calculate the byte location where the tile map starts, shift the a (address) bits left by 11 (multiply by 2048.) The SC size is the dimensions of the tile map; if using 8x8 tile mode, this allows BG dimensions of 256 or 512 pixels; if in 16x16 mode, the dimensions can be 512 or 1024 pixels. Note that, since there is only 64K of VRAM, the most significant bit must be zero.

When using a screen size wider than 32 tiles, the format is a little different than you might expect. When the width is 64 tiles, then rather than each line in the tile map extending to 128 bytes (instead of 64), there will actually be two tile maps, stored one right after the other in memory. The first tile map will contain the left 32 tiles (x coordinates 0 to 255, when using 8x8 tiles), and the next tile map will contain the right 32 tiles (x coordinates 256 to 511, when using 8x8 tiles. Setting the scroll register to 512, then, will be the same at setting it to zero.)

A note about using 16x16 tiles: These are stored in exactly the same way as 16x16 sprites; that is, the first and second rows have 14 ignored tiles between them.

[Back to top](#)

BG Character Data Area Designation

```
$210B  wb++?- BG12NBA
$210C  wb++?- BG34NBA
        ddddcxxx bbbbaaaa
        dddd = BG4 Tile Base Address (in 4K-word steps)
        cxxx = BG3 Tile Base Address (in 4K-word steps)
        bbbb = BG2 Tile Base Address (in 4K-word steps)
        aaaa = BG1 Tile Base Address (in 4K-word steps)
```

This register selects the location in VRAM where the tile map starts. The byte address is calculated by shifting the four bits left by 13 (multiplying by 8192). Simply spoken: Saving “\$63” into \$210B makes the PPU look for the Tileset for BG2 at \$6000 in the VRAM and for BG1 at \$3000. Note that, since there is only 64K of VRAM, the highest of the four bits must be set to 0. Ignored in Mode 7 (Base is always zero).

BG1 and Mode 7 Scroll

```
$210D  ww++++- BG1H0FS
        ww++++- M7H0FS
$210E  ww++++- BG1V0FS
        ww++++- M7V0FS
        -----xx xxxxxxxx
        ---mmmmm mmmmmmmmm
        xxxxxxxxxx      = The BG offset, 10 bits
        mmmmmmmmmmmmm = The Mode 7 BG offset, 13 bits two's-complement
signed
```

These are actually two registers in one (or would that be “4 registers in 2”?). Anyway, writing \$210D will write both BG1H0FS which works exactly like the rest of the BGnxOFS registers below (\$210F-\$2114), and M7H0FS which works with the M7* registers (\$211B-\$2120) instead.

Modes 0-6 use BG1xOFS and ignore M7xOFS, while Mode 7 uses M7xOFS and ignores BG1H0FS. See the appropriate sections below for details, and note the different formulas for BG1H0FS versus M7H0FS.

[Back to top](#)

BG2, BG3 and BG4 Scroll

```
$210F  ww++++- BG2H0FS
$2110  ww++++- BG2V0FS
$2111  ww++++- BG3H0FS
$2112  ww++++- BG3V0FS
$2113  ww++++- BG4H0FS
$2114  ww++++- BG4V0FS
```

```
-----XX XXXXXXXX
xxxxxxxxxx = The BG offset, 10 bits
```

Note that these are “write twice” registers, first the low byte is written then the high. Current theory is that writes to the register work like this:

```
BGnH0FS = (Current<<8) | (Prev&~7) | ((Reg>>8)&7);
Prev = Current;
or
BGnV0FS = (Current<<8) | Prev;
Prev = Current;
```

Note that there is only one Prev shared by all the BGnXOFS registers. This is NOT shared with the M7* registers (not even M7xOFS and BG1xOFS).

Also, note that all BGs wrap if you try to go past their edges (if a pixel value is placed in this register that is larger than the width of the BG, a modulus can be performed to determine what the actual pixel will be that is displayed. For example, if the BG1 horizontal pixel value is set to 257, but the width of the BG is 256 pixels, the result will be the same as if it was set to 1). Thus, the maximum offset value in BG Modes 0-6 is 1023, since you have at most 64 tiles (if x/y of BGnSC is set) of 16 pixels each (if the appropriate bit of BGMODE is set).

Horizontal scrolling scrolls in units of full pixels no matter if we’re rendering a 256-pixel wide screen or a 512-half-pixel wide screen. However, vertical scrolling will move in half-line increments if interlace mode is active.

[Back to top](#)

Video Port Control

```
$2115 wb++?- VMAIN
i---ttrr
i = Address increment mode:
    0 => increment after writing to $2118/reading from $2139
    1 => increment after writing to $2119/reading from $213A
tt = Address translation
    00 = No translation
    01 = 8bit rotate
    10 = 9bit rotate
    11 = 10bit rotate
rr = Address increment amount
    00 = Normal increment by 1
    01 = Increment by 32
    10 = Increment by 128
    11 = Increment by 128
```

This register controls the way data is uploaded to VRAM. The bits in here are a bit weird, but can be useful. When you want to change only the high byte of a series of VRAM locations (register $\$2116 * 2 + 1$), you should set i to 1. When you want to change just the low byte, set i to 0. When you want to

write a whole word, you should set *i* to 0; otherwise, if *i*=1, writing a word will cause the high byte of the first location to be changed, followed by the low byte of the next location.

The address translation (*tt*) is intended for bitmap graphics (where one would have filled the BG Map by increasing Tile numbers), technically it does thrice left-rotate the lower 8, 9, or 10 bits of the Word-address. As an example if \$2116-\$2117 are set to #0003, then word address #0018 will be written instead, and \$2116-\$2117 will be incremented to 0004:

| Translation | Bitmap Type | Port [2116h/17h] | VRAM Word-Address |
|--------------|--------------------------|-------------------|---------------------|
| 8bit rotate | 4-color; 1 word/plane | aaaaaaaaYYYxxxxx | --> aaaaaaaxxxxYYY |
| 9bit rotate | 16-color; 2 words/plane | aaaaaaaaYYYxxxxxP | --> aaaaaaaxxxxPYYY |
| 10bit rotate | 256-color; 4 words/plane | aaaaaaYYYxxxxxPP | --> aaaaaaxxxxPPYYY |

Where “aaaaa” would be the normal address MSBs, “YYY” is the Y-index (within a 8×8 tile), “xxxxx” selects one of the 32 tiles per line, “PP” is the bit-plane index (for BGs with more than one Word per plane). For the intended result (writing rows of 256 pixels) the Translation should be combined with Increment Step=1.

For Mode 7 bitmaps one could eventually combine step 32/128 with 8bit/10bit rotate:

```
8bit-rotate/step32  aaaaaaaaaXXXxxYYY --> aaaaaaaxxYYYXXX
10bit-rotate/step128 aaaaaaXXXxxxxYYY --> aaaaaaxxxxYYYXXX
```

Though the SNES can't access enough VRAM for fullscreen Mode 7 bitmaps. Step 32 (without translation) is useful for updating BG Map columns (eg. after horizontal scrolling).

[Back to top](#)

VRAM Address

```
$2116  wl++?- VMADDL
$2117  wh++?- VMADDH
        aaaaaaaa aaaaaaaa = Word address for accessing VRAM
```

VRAM Address for reading/writing. This is a WORD address (2-byte steps), the PPU could theoretically address up to 64K-words (128K-bytes), in practice, only 32K-words (64K-bytes) are installed in SNES consoles (VRAM address bit15 is not connected, so addresses 8000h-FFFFh are mirrors of 0-7FFFh).

When reading from VRAM, a “dummy read” must be performed after writing to this register; the first value read is supposed to be meaningless. No “dummy write” is required, however.

After reading/writing VRAM Data, the Word-address can be automatically incremented by 1,32,128 (depending on the Increment Mode in Register \$2115) (Note: the Address Translation feature is applied only “temporarily” upon memory accesses, it doesn't affect the value in Register \$2116-\$2117). Writing to \$2116-\$2117 does prefetch 16bit data from the new address (for later reading).

[Back to top](#)

VRAM Data Write

```
$2118  wl++-- VMDATAL
$2119  wh++-- VMDATAH
        xxxxxxxx xxxxxxxx = Data to write to VRAM
```

This writes data to VRAM. The writes take effect immediately, even if no increment is performed. The address is incremented when one of the two bytes is written; which one depends on the setting of bit 7 of register \$2115. Depending on the Increment Mode the address does (or doesn't) get automatically incremented after the write. Keep in mind the address translation bits of \$2115 as well. The interaction between these registers and \$2139-\$213A is unknown.

[Back to top](#)

Mode 7 Settings

```
wb++?- M7SEL
        rr---yx
        rr = Screen Over
                00 = Wrap within 128x128 tile area
                01 = Wrap within 128x128 tile area (same as 0)
                10 = Outside 128x128 tile area is Transparent
                11 = Outside 128x128 tile area is filled by Tile $00
        y  = Screen V-Flip (0=Normal, 1=Flipped) (flip 256x256 "screen")
        x  = Screen H-Flip (0=Normal, 1=Flipped) (flip 256x256 "screen")
```

[Back to top](#)

Mode 7 Matrix A, B , C and D

```
$211B  ww++++ M7A (and Maths 16bit operand)
$211C  ww++++ M7B (and Maths 8bit operand)
$211D  ww++++ M7C
$211E  ww++++ M7D
        aaaaaaaaaa aaaaaaaaaa = Signed 16bit values in 1/256 pixel units (1bit
sign, 7bit integer, 8bit fraction)
```

Note that these are “write twice” registers, first the low byte is written then the high. Current theory is that writes to the register work like this:

```
Reg = (Current<<8) | Prev;
Prev = Current;
```

Note that there is only one Prev shared by all these registers. This Prev is NOT shared with the BGnxOFS registers, but it IS shared with the M7xOFS registers. These set the matrix parameters for

Mode 7. The values are an 8-bit fixed point, i.e. the value should be divided by 256.0 when used in calculations. See below for more explanation.

The product $A*(B \gg 8)$ may be read from registers \$2134-\$2136. There is supposedly no important delay. It may not be operative during Mode 7 rendering.

[Back to top](#)

Mode 7 Center X and Y

```
$211F  ww++++- M7X
$2120  ww++++- M7Y
      ---xxxxx xxxxxxxx = Signed 13bit values in pixel units (1bit sign,
      12bit integer, 0bit fraction)
```

Note that these are “write twice” registers, like the other M7* registers. See above for the write semantics. The value is 13 bit two’s-complement signed. The matrix transformation formula is:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} * \begin{bmatrix} SX + M7H0FS - CX \\ SY + M7V0FS - CY \end{bmatrix} + \begin{bmatrix} CX \\ CY \end{bmatrix}$$

Note: SX/SY are screen coordinates. X/Y are coordinates in the playing field from which the pixel is taken. If \$211A bit 7 is clear, the result is then restricted to $0 \leq X \leq 1023$ and $0 \leq Y \leq 1023$. If \$211A bits 6 and 7 are both set and X or Y is less than 0 or greater than 1023, use the low 3 bits of each to choose the pixel from character 0. The bit-accurate formula seems to be something along the lines of:

```
#define CLIP(a) (((a)&0x2000)?((a)|~0x3ff):((a)&0x3ff))
```

```
X[0,y] = ((A*CLIP(H0FS-CX))&~63)
        + ((B*y)&~63) + ((B*CLIP(V0FS-CY))&~63)
        + (CX<<8)
```

```
Y[0,y] = ((C*CLIP(H0FS-CX))&~63)
        + ((D*y)&~63) + ((D*CLIP(V0FS-CY))&~63)
        + (CY<<8)
```

```
X[x,y] = X[x-1,y] + A
Y[x,y] = Y[x-1,y] + C
```

(In all cases, X[] and Y[] are fixed point with 8 bits of fraction)

[Back to top](#)

CGRAM Address

```
$2121  wb++++- CGADD
```

aaaaaaaa = CGRAM word address

This sets the word address (byte address * 2, i.e. color) to begin uploading (or downloading) data to CGRAM, which will be affected by \$2122 and \$213B.

Writing "0" to \$2121 will change the "currently selected color index" used by \$2122, to 0. Upon writing a color to \$2122, the color will be stored into the array index selected by \$2121, which in this case would be 0 - if you wrote 0 to \$2121 before writing a color to \$2122.

Keep in mind the color index accessed by \$2121 will automatically increment by 1 after writing a color to \$2122. This is an effect generated by \$2122 after being used in case you want to write specific colors in a series.

[Back to top](#)

CGRAM Data Write

```
$2122  ww+++- CGDATA
        bbbbbbbb = byte to write to CGRAM
        Writes to EVEN and ODD byte-addresses work as follows:
        Write to EVEN address --> set Cgram_Lsb = Data    ;memorize
value
        Write to ODD address  --> set WORD[addr-1] = Data*256 +
Cgram_Lsb
```

This register writes a byte to CGRAM. After the byte is stored, the CGRAM address is incremented so that the next write or read will be to the following byte. Accesses to CGRAM are handled just like accesses to the low table of OAM, see \$2104 for details. Note that the color values are stored in BGR order (-bbbbbbgg ggrrrrr).

[Back to top](#)

Window Mask Settings

```
$2123  wb+++- W12SEL - Window Mask Settings for BG1 and BG2
$2124  wb+++- W34SEL - Window Mask Settings for BG3 and BG4
$2125  wb+++- WOBJSEL - Window Mask Settings for OBJ and Color Window
        aabbccdd
        2123h 2124h 2125h
        aa = BG2   BG4   MATH  Window-2 Area (0..1=Disable, 1=Inside,
2=Outside)
        bb = BG2   BG4   MATH  Window-1 Area (0..1=Disable, 1=Inside,
2=Outside)
        cc = BG1   BG3   OBJ   Window-2 Area (0..1=Disable, 1=Inside,
2=Outside)
        dd = BG1   BG3   OBJ   Window-1 Area (0..1=Disable, 1=Inside,
2=Outside)
```


Allows to select if the window area is inside or outside the X1,X2 coordinates, or to disable the area. In other words, these registers determine which Windows to apply to which BGs, sprite (OBJ) or color window (MATH), and whether clipping should be performed inside or outside the window. To enable windowing, the appropriate bits in registers \$212E and \$212F must be set in addition to the bits in these registers.

[Back to top](#)

Window Position

```
$2126  wb+++ - WH0 - Window 1 Left Position
$2127  wb+++ - WH1 - Window 1 Right Position
$2128  wb+++ - WH2 - Window 2 Left Position
$2129  wb+++ - WH3 - Window 2 Right Position
        xxxxxxx = Window Position ($00..$FF; 0=leftmost, 255=rightmost)
```

Specifies the horizontal boundaries of the windows. Note that there are no vertical boundaries (these could be implemented by manipulating the window registers via IRQ and/or HDMA). The “inside-window” region extends from X1 to X2 (that, including the X1 and X2 coordinates), so the window width is X2-X1+1. If the width is zero (or negative), then the “inside-window” becomes empty, and the whole screen will be treated “outside-window”.

[Back to top](#)

Window Mask Logic

```
$212A  wb+++ - WBGL0G - Window mask logic for BGs
        44332211
$212B  wb+++ - WOBJLOG - Window mask logic for OBJs and Color Window
        ---mmoo
        44 = BG4 Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
        33 = BG3 Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
        22 = BG2 Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
        11 = BG1 Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
        mm = MATH Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
        oo = OBJ Window 1/2 Mask Logic (0=OR, 1=AND, 2=XOR, 3=XNOR)
```

Allows to merge the Window 1 and 2 areas into a single “final” window area (which is then used by TMW, TSW, and CGWSEL). The OR/AND/XOR/XNOR logic is applied ONLY if BOTH window 1 and 2 are enabled (in WxxSEL registers). If only one window is enabled, then that window is used as is as “final” area. If both are disabled, then the “final” area will be empty. Note: “XNOR” means “1 XOR area1 XOR area2” (ie. the inverse of the normal XOR result).

From:
<https://www.ff6hacking.com/wiki/> - **ff6hacking.com** wiki

Permanent link:
<https://www.ff6hacking.com/wiki/doku.php?id=ff3:ff3us:doc:snes:register&rev=1564801083>

Last update: **2019/08/03 02:58**

