Xevious 68K OSD Interface Specification v0.6

Video Layers

Background Layer

The background layer consists of 8x8 pixel tiles, with 4 colours per pixel (no transparency). Each tile has a 7-bit colour attribute which is used to index into a table of 4-colour CLUTs. Tiles can be flipped on X and/or Y axes. The background layer is the bottom-most layer.

Background attribute (colour) and tile (video) RAM are $800 bytes each arranged as 64 rows of 32 tiles as viewed on the vertical monitor. The visible display area is 36 rows of 28 tiles. The background layer can be scrolled vertically and layer wraps at the first and last lines.

There are 512 background tiles in total (2 banks of 256 tiles). The tile (0-255) is specified in the video RAM, and the bank is specified in the attribute (colour) RAM.

[Mapping TBD.]

The background layer is the game map, and is only written one row at a time in a single routine in the SUB program. The row is constructed above the visible display, so there is no need to synchronise with the VBLANK.

Foreground Layer

The foreground layer consists of 8x8 pixel tiles, with just 1 colour per pixel or transparency. Each tile has a 6-bit colour attribute which is used to index into the colour table directly. Tiles can be flipped on X and/or Y axes. The foreground layer is the foremost layer (including sprites).

Foreground attribute (colour) and tile (video) RAM are $800 bytes each arranged as 64 rows of 32 tiles as views on the vertical monitor. The visible display area is 36 rows of 28 tiles. The foreground layer cannot be scrolled, and so only the latter 36 rows of the upper 40 rows are ever visible.

Although there are two banks of 256 foreground tiles in the graphics ROM, the 2nd bank comprises X-flipped versions of the 1st bank, and are only used in cocktail mode. The 2nd bank can only be selected by the video hardware.

[Mapping TBD.]

The foreground layer is the overlay text, and is written in many places in both the MAIN and SUB programs. There is no need to synchronise with the VBLANK.

Sprites

Xevious has 64 sprites consisting of 16x16 tiles, with 8 colours per pixel, which includes transparency. Each tile has a 7-bit colour attribute which is used to index into a table of 8-colour LUTs. Tiles can be flipped on X and/or Y axes.

Sprites can also be joined to form double height and/or width sprites. These sprites use contiguous tile codes to form the composite sprite.

Sprite registers are split into three (3) banks of 64 2-byte registers mapped into CPU memory space.

Sprite hardware registers are updated (ideally) during VBLANK. The MAIN program updates the first 32 sprites during the VBLANK ISR; the SUB CPU updates the latter 32 sprites as soon as the MAIN CPU has updated its sprites (which is not guaranteed to conclude during VBLANK).

OSD Interface

osd_enable_vblank_interrupt

Enables the VBLANK interrupt on the host platform.

Called from the Xevious main program after data areas have been initialised.

osd_ack_vblank_interrupt

ACKs the VBLANK interrupt on the host platform.

Called from the Xevious MAIN program to clear pending VBLANK interrupts before they are enabled.

osd_disable_vblank_interrupt

Disables the VBLANK interrupt on the host platform.

Currently called from the Xevious SUB program when updating sprite hardware registers. This is because VRAM accesses on the Neo Geo rely on an address register and therefore are not re-entrant.

osd_w_fg_colourram

Writes directly to the foreground tilemap layer colour (attribute) RAM.

The OSD layer should directly update the attributes (colour and flip) of the foreground tile at this offset. Generally called from a non-interrupt context.

On entry:	
D0.W	Byte offset in foreground colourram memory
D1.B	Data to write

Bit(s)	**Description**
7	Flip Y
6	Flip X
5:2	Colour 3:0
1:0	Colour 5:4

Preserved: D0-D3/A0/A1

osd_w_bg_colourram

Writes directly to the background tilemap layer colour (attribute) RAM.

The OSD layer should directly update both the attribute and code of the background tile at this offset. The code must also be updated because it is derived from bit 0 of the attribute.

On entry	
D0.W	Byte offset in background colourram memory
D1.B	Data to write

Bit(s)	**Description**
7	Flip Y
6	Flip X
5:2	Colour 3:0
1:0	Colour 6:5, and bit 0 is also bit 8 of the tile code

Note that bit 4 of the colour is derived from bit 7 of the videoram byte.

Preserved: D0-D4/A0/A1

osd_w_fg_videoram

Writes directly to the foreground tilemap layer video RAM.

The OSD layer should directly update the tile at this offset.

On entry	
D0.W	Byte offset in foreground videoram memory
D1.B	Data to write

Preserved: D0-D3/A0/A1

osd_w_bg_videoram

Writes directly to the background tilemap layer video RAM.

The OSD layer should directly update the both the attribute and code of the background tile at this offset. The attribute must also be updated because the colour is derived from bit 7 of the code.

On entry	
D0.W	Byte offset in background videoram memory
D1.B	Data to write

Note that bit 8 of the code is derived from bit 0 of the attribute.

Preserved: D0-D3/A0/A1

osd_update_scroll_hw

Updates the scroll position of the background tilemap layer.

The OSD layer should directly scroll the background layer.

[Further explanation TBD]

Xevious calls this from the SUB program, immediately after the SUB program has updated the latter half of the sprite hardware registers. It should be done during VBLANK, but it is not guaranteed by the programming.

On entry	
D6.W	Scroll register value ($0-$1ff)

osd_update_sprite_shadow

Updates the shadow copy of the sprite hardware registers. This routine should iterate through all 64 entries in the object table (a5) and set sprite register values accordingly. It is also responsible for deactivating sprites in both the object table and on-screen.

This is called from the SUB program, as the last function before spinning on the next VBLANK. The idea is to pre-format the sprite hardware register data so that the ISR need only require a block copy to update the sprites on-screen.

It is necessary to update a few object table values when sprites are no longer visible.

The following table outlines the actions required depending on the STATE of the object:

Object state	**Action(s)**
0	Flagged to be made inactive. 1. Zero X & Y members of the object table 2. Set STATE=1 (inactive) in object table 3. Deactivate/hide sprite on target platform
1	Inactive. No further action required
n	Active. Update sprite registers from object data structure

Note that whilst shadow registers are in an OSD-specific format, Xevious uses these values for hit-box calculations. The registers also potentially need tweaking (on the Neo Geo at least) to support double height/width sprites, which breaks these calculations.

Therefore, the code must maintain two shadow copies of sprite registers, at least for X and Y coordinates; the 2nd copy in a platform-agnostic (raw) format to facilitate the calculations.

The 2nd copy of register values (byte) are stored at **sprite_shadow_msb** and must be updated by this routine. The format of the array (64 sprites) is as follows:

Offset	**Description**
0	Sprite 0: ($EF - Y[7:0])
1	Sprite 0: ((X[8:0] + 8) >> 1)
(2n)	Sprite <n>: ($EF - Y[7:0])
(2n+1)	Sprite <n>: ((X[8:0] + 8) >> 1)

On entry	
A5.L	Object table address

Member	Description
_STATE.b	Object state. 1 = do not update shadow 0 = deactivate sprite h/w
_X.w	Sprite X[8:0] in word [13:5]
_Y.w	Sprite Y[7:0] in word [12:5]
_ATTR.b	bit 7 = tile bank (0/1) bit 3 = yflip bit 2 = xflip bit 1 = double height bit 0 = double width
_CODE.b	tile code ($00-$ff)
_COLOUR.b	Colour in word [6:0]

For double height and/or double width sprites, the tile code is as follows:

Normal	code=code	code+0	
Double width	code=code&(~1)	code+0	
		code+1	
Double height	code=code&(~2)	code+2	code+0
Double width, height	code=code&(~3)	code+2	code+0
		code+3	code+1

osd_update_32_sprite_hw

Updates a block of 32 sprite hardware registers from the shadow copy. Xevious calls this twice; once from the MAIN CPU VBLANK ISR for the 1st 32 sprites (ground-based objects and Bacura), and once from the SUB CPU main loop (as soon as the VBLANK semaphore has been triggered) for the latter 32 sprites (Solvalou, bombs, bullets and flying enemies).

This is executing during the VBLANK ISR and so should be as efficient as possible.

On entry	
D0.W	Base (starting) sprite number

osd_read_dipswitches

Returns the current state of dipswitches A & B.

On exit DSWA (d0)

Bit(s)	**Description**
7	Cabinet Upright/Cocktail (ignored)
6:5	Lives **Value** / **Description** 11 / 3 10 / 2 01 / 1 00 / 5
4:2	Bonus Life **Value** / **Lives=1/3/5** / **Lives=5** 111 / 20K, 60K, every 60K / 20K, 70K, every 70K 110 / 10K, 40K, every 40K / 10K, 50K, every 50K 101 / 10K, 50K, every 50K / 20K, 50K, every 50K 100 / 20K, 50K, every 50K / 20K, 60K, every 60K 011 / 20K, 70K , every 70K / 20K, 80K, every 80K 010 / 20K, 80K, every 80K / 30K, 100K, every 100K 001 / 20K, 60K only / 20K, 80K only 000 / None / None
1:0	Coinage (ignored)

<table>
<tr><th colspan="2">On exit DSWB (d1)</th></tr>
<tr><th>Bit(s)</th><th>Description</th></tr>
<tr><td>7</td><td>Freeze (ignored)</td></tr>
<tr><td>6:5</td><td>Difficulty
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>11</td><td>Normal</td></tr>
<tr><td>10</td><td>Easy</td></tr>
<tr><td>01</td><td>Hard</td></tr>
<tr><td>00</td><td>Hardest</td></tr>
</table></td></tr>
<tr><td>4</td><td>Button 2 – ACTIVE LOW</td></tr>
<tr><td>3:1</td><td>Coinage (ignored)</td></tr>
<tr><td>0</td><td>Flags Award Bonus Life
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>Yes</td></tr>
<tr><td>0</td><td>No</td></tr>
</table></td></tr>
</table>

osd_read_p1_inputs

Returns P1 joystick controls and Button 1 (shoot). Note that button 2 (bomb) is returned via reading the dipswitches (above).

The inputs are handled by the Namco custom co-processors and the values returned (d0) are ordinals representing only valid combinations of joystick positions.

On exit (d0)	
Value	**Joystick Direction**
$F0	UP
$F1	RIGHT+UP
$F2	RIGHT
$F3	RIGHT+DOWN
$F4	DOWN
$F5	LEFT+DOWN
$F6	LEFT
$F7	LEFT+UP
$F8	NONE/INVALID

Additionally, bits 5 and 4 (active low) both represent the Button 1 as follows:

On exit (d0)	
Bit	**Description**
5	Button 1
4	Button 1 pressed/down (one-shot)

Bit 4 is used on the high score name entry screen.

osd_read_p2_inputs

Returns P2 joystick controls and Button 1 (shoot). Note that Button 2 (bomb) is returned via reading the dipswitches (above).

The returned data (d0) is the same as ***osd_read_p1_inputs***. On most platforms it may be adequate to simply branch to ***osd_read_p1_inputs*** as Xevious has no simultaneous 2-player mode.

osd_read_coin

Reads the state of a new coin insert (one-shot).

On exit (d0)	
Value	**Description**
0	No new coin inserted
non-zero	New coin inserted (one-shot)

Currently it is also required to set the Z flag accordingly.

osd_read_start

Reads start button states.

On exit (d0)	
Bit	**Description**
1	2 Player START
0	1 Player START

osd_debug_hook

This function is used to assist in debugging. It has no pre-defined inputs, outputs or function. The idea is to change the function to whatever is required at the time, and insert function calls in the source wherever required at the time.

It exists to mitigate the requirement to add a new global function to the project.