ppu.h File Reference

User Library for the FP-GAme PPU. More...

```
#include <stdlib.h>
#include <stdint.h>
#include <sys/types.h>
```

Go to the source code of this file.

Classes

struct	**pattern_t** A single 8x8 tiles worth of pattern data (8 rows of 8 pixels at 4bpp) (32B of data) More...
struct	**palette_t** A palette which contains 15 colors. More...
struct	**sprite_t** A sprite. More...

Typedefs

typedef unsigned	**pattern_addr_t** Address into pattern memory. Generate using **ppu_pattern_addr**.
typedef uint16_t	**tile_t** Tile data representation (technically 2B of data)

Enumerations

enum	**layer_e** { **LAYER_BG** = 1 , **LAYER_FG** = 2 , **LAYER_SPR** = 4 } An enum which specifies a render layer. More...
enum	**mirror_e** { **MIRROR_NONE** = 0 , **MIRROR_X** = 1 , **MIRROR_Y** = 2 , **MIRROR_XY** = 3 } Mirror state for graphics. More...
enum	**render_prio_e** { **PRIO_IN_BACK** = 0 , **PRIO_IN_MIDDLE** = 1 , **PRIO_IN_FRONT** = 2 } Rendering priority for sprites. More...

Functions

int	**ppu_enable** (void) Enable the PPU. More...
void	**ppu_disable** (void) Disable the PPU. More...
int	**ppu_update** (void) Request for the current frame changes to be send to the PPU on the next available frame. More...
int	**ppu_write_vram** (const void *buf, size_t len, off_t offset) Write directly to the VRAM buffer. More...
pattern_addr_t	**ppu_pattern_addr** (unsigned pattern_id, unsigned x, unsigned y) Generates a pattern_addr_t using a pattern_id, and relative (x, y) position. More...
tile_t	**ppu_make_tile** (**pattern_addr_t** pattern_addr, unsigned palette_id, **mirror_e** mirror) Generate a tile data for use with the ppu_write_tile functions. More...
void	**ppu_make_sprite** (**sprite_t** *sprite, **pattern_addr_t** pattern_addr, unsigned width, unsigned height, unsigned palette_id, **render_prio_e** prio, **mirror_e** mirror)

	Generate a sprite data for use with the ppu_write_sprites function. More...
void	**ppu_load_tilemap** (**tile_t** *tilemap, unsigned len, char *file) Loads tile-data from a file into an linear array of tile_t. More...
void	**ppu_load_pattern** (**pattern_t** *pattern, char *file) Loads an 8x8-pixel tile graphic into `pattern` from `file`. More...
void	**ppu_load_palette** (**palette_t** *palette, char *file) Loads a color palette from `file` into `palette`. More...
int	**ppu_write_tiles_horizontal** (**tile_t** *tiles, unsigned len, unsigned x_i, unsigned y_i, unsigned count) Writes an array to a horizontal segment of tiles. More...
int	**ppu_write_tiles_vertical** (**tile_t** *tiles, unsigned len, unsigned x_i, unsigned y_i, unsigned count) Writes an array to a vertical segment of tiles. More...
int	**ppu_write_pattern** (**pattern_t** *pattern, unsigned width, unsigned height, **pattern_addr_t** pattern_addr) Writes **pattern_t** patterns (8x8-pixel tiles) to a specified location in Pattern RAM. More...
int	**ppu_write_palette** (**palette_t** *palette, **layer_e** layer_id, unsigned palette_id) Overwrites a palette in Palette RAM. More...
int	**ppu_write_sprites** (**sprite_t** *sprites, unsigned len, unsigned sprite_id_i) Overwrites one or more sprite data entries in Sprite RAM. More...
int	**ppu_set_bgcolor** (unsigned color) Set the universal background color of the PPU. More...
int	**ppu_set_scroll** (**layer_e** tile_layer, unsigned scroll_x, unsigned scroll_y) Set pixel scroll of the background or foreground tile layer. More...
int	**ppu_set_layer_enable** (unsigned enable_mask) Enable or disable one or more of the three PPU render layers using a bit-mask. More...

Detailed Description

User Library for the FP-GAme PPU.

Author

Joseph Yankel

Attention

Modifications to the PPU will not be accepted during certain busy states managed by the Kernel. Any functions which attempt to modify PPU data will return -1 if the modification could not be made. You are encouraged to poll these functions until they return 0 (success) if you want to ensure your changes are made.

Invalid arguments (see the function's documentation) will result in a console warning and exiting of the program. This is to help you (the user) find bugs and unwanted behaviours.

Enumeration Type Documentation

◆ layer_e

enum **layer_e**

An enum which specifies a render layer.

These enum entries can be ORd together to form a bitmask.

See also

ppu_set_layer_enable.

Enumerator	
LAYER_BG	Denotes the background tile render layer.
LAYER_FG	Denotes the foreground tile render layer.
LAYER_SPR	Denotes the sprite render layer.

◆ mirror_e

enum **mirror_e**

Mirror state for graphics.

Enumerator	
MIRROR_NONE	Pattern is not mirrored.
MIRROR_X	Pattern is horizontally flipped.
MIRROR_Y	Pattern is vertically flipped.
MIRROR_XY	Pattern is both horizontally and vertically flipped.

◆ render_prio_e

enum **render_prio_e**

Rendering priority for sprites.

Enumerator	
PRIO_IN_BACK	Sprite apperas behind both background and foreground tile layers.
PRIO_IN_MIDDLE	Sprite appears in front of background but behind foreground tile layer.
PRIO_IN_FRONT	Sprite appears in front of background and foreground tile layers.

Function Documentation

◆ ppu_disable()

void ppu_disable (void)

Disable the PPU.

Releases the lock on the PPU. Other processes will be able to reserve access to the PPU.

It is illegal to call this function if the PPU is not currently enabled and owned by the calling process.

◆ ppu_enable()

int ppu_enable (void)

Enable the PPU.

Attempts to lock PPU access to this process. If successful, only this process will be able to write to the PPU.

Fails if the PPU is already owned by another process.

The caller of this function must call ppu_disable before program exit to prevent resource leaks.

Returns

0 on success; -1 on error

◆ ppu_load_palette()

void ppu_load_palette (**palette_t** * palette,
char * file
)

Loads a color palette from `file` into `palette`.

`file` must be a simple text file, containing 15 lines, each with a 24-bit hex color. For example, each line has something of the form: FF0000 The example above is a hex representation for the color RED.

Parameters

palette Palette instance to copy the color data from `file` to.

file File path of the text file to copy color data from.

◆ ppu_load_pattern()

```
void ppu_load_pattern ( pattern_t *  pattern,
                        char *       file
                      )
```

Loads an 8x8-pixel tile graphic into `pattern` from `file`.

`file` must be a simple text file, containing 8 lines, each with 8 hex chars: 0-F. Each hex char represents a pixel's color for any palette. 0 is always transparent, 1-F correspond to the 15 available colors in a palette.

Example: 112233445 F00000005 F00000006 E00000006 E00000007 D00000007 D00000008 CCBBAA998

A test file with the above text represents a multi-colored box outline with transparent center.

Parameters

pattern Pattern instance to copy the pixel data from `file` to.

file File path of the text file to copy pixel data from.

◆ ppu_load_tilemap()

```
void ppu_load_tilemap ( tile_t *  tilemap,
                        unsigned  len,
                        char *    file
                      )
```

Loads tile-data from a file into an linear array of tile_t.

The text file contains tile_t entries specified in the following format:

- An entry is formatted like (XXX,X,X), where X is a hex number (0-F).
- The first three hex numbers are the pattern ID for this tile. It ranges from 000 to 3FF.
- The second entry is the palette ID to use for this tile. It ranges from 0 to F.
- The last entry are the tile mirroring bits. This value ranges from 0 to 3, where:
  - 0 -> No mirror
  - 1 -> Horizontal Mirroring
  - 2 -> Vertical Mirroring
  - 3 -> Both Horizontal AND Vertical Mirroring
- Each entry is separated either by a space or by a newline.

Parameters

tilemap Array of tile_t to load into.

len Number of tiles to copy to `tilemap`.

file Path of the text file to open and read from.

◆ ppu_make_sprite()

```
void ppu_make_sprite ( sprite_t *       sprite,
                       pattern_addr_t   pattern_addr,
                       unsigned         width,
                       unsigned         height,
                       unsigned         palette_id,
                       render_prio_e    prio,
                       mirror_e         mirror
                     )
```

Generate a sprite data for use with the ppu_write_sprites function.

Importantly, the sprite's position is defaulted to (0, 0).

Parameters

sprite	Pointer to an empty **sprite_t** struct you want to initialize.
pattern_addr	The address into Pattern RAM where this sprite starts. Recommended to generate using **ppu_pattern_addr**.
width	Horizontal width of sprite in tiles. Can take values in [1, 4].
height	Vertical height of sprite in tiles. Can take values in [1, 4].
palette_id	Palette from the sprites section of Palette RAM to use. Must be within [0, 31].
prio	Render priority for this sprite.
mirror	Horizontal/Vertical mirror setting for this sprite.

◆ ppu_make_tile()

```
tile_t ppu_make_tile ( pattern_addr_t   pattern_addr,
                       unsigned         palette_id,
                       mirror_e         mirror
                     )
```

Generate a tile data for use with the ppu_write_tile functions.

Parameters

pattern_addr	The address of this tile's pattern in Pattern RAM (see **ppu_pattern_addr**)
palette_id	The numerical id of the palette (location in Palette RAM) this tile will use. This must be within range [0, 16]. The final palette comes from the background layer palette section of Palette RAM if this tile is applied to the background tile layer, and similarly for forground palettes.
mirror	Mirror state for this tile's pattern.

Returns

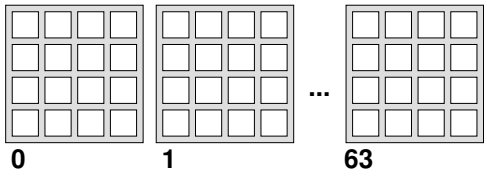
A tile_t representing the tile data formed by the inputs.

◆ ppu_pattern_addr()

```
pattern_addr_t ppu_pattern_addr ( unsigned pattern_id,
                                  unsigned x,
                                  unsigned y
                                )
```

Generates a pattern_addr_t using a pattern_id, and relative (x, y) position.

Pattern RAM is organized into blocks of 32x32-pixel chunks (or 4x4 8x8-pixel tile groups).



Note each small square tile in the diagram above represents an 8 pixel by 8 pixel tile. There are 16 such tiles per tile group.

`pattern_id` selects which of these tile groups to write to. (x, y) indicates a position within this 16 tile group with the origin (0, 0) at the top left.

Parameters

pattern_id Index into Pattern RAM selecting a tile group. Range [0, 63].

x Within the selected tile group, horizontal tile offset. Range [0, 3].

y Within the selected tile group, vertical tile offset. Range [0, 3].

Returns

A pattern_addr_t representing the address into Pattern RAM formed by the arguments.

◆ ppu_set_bgcolor()

int ppu_set_bgcolor (unsigned color)

Set the universal background color of the PPU.

The universal background color is the color displayed when all PPU render layers are transparent.

This function will set this color to be displayed at the next **ppu_update()**.

Remarks

Any higher-order bits [31:24] in `color` will be ignored!

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

color 32-bit color holding a 24-bit RRGGBB hex color value. For example, 0xFF0000 for red.

Returns

0 on success; -1 if PPU busy

◆ ppu_set_layer_enable()

int ppu_set_layer_enable (unsigned enable_mask)

Enable or disable one or more of the three PPU render layers using a bit-mask.

The enable mask has three bits which enable or disable the PPU render layers as follows: Bit 0: Enable (1) or disable (0) the background tile layer Bit 1: Enable (1) or disable (0) the foreground tile layer Bit 2: Enable (1) or disable (0) the sprite layer To generate the enable_mask, use an OR of the layer_e options. For example, to enable both tile layers and disable the sprite layer set enable_mask = BG | FG.

Call this function before a **ppu_update()** to ensure the layer will be enabled on the next frame.

Remarks

Any higher-order bits in enable_mask not specified above will be ignored!

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

enable_mask Bit-mask used to enable/disable PPU rendering layers.

Returns

0 on success; -1 if PPU busy

◆ ppu_set_scroll()

```
int ppu_set_scroll ( layer_e  tile_layer,
                     unsigned scroll_x,
                     unsigned scroll_y
                   )
```

Set pixel scroll of the background or foreground tile layer.

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

tile_layer Either LAYER_BG or LAYER_FG. LAYER_SPR doesn't support layer scrolling.

scroll_x Horizontal pixel scroll. Values must be [0, 511].

scroll_y Vertical pixel scroll. Values must be [0, 511].

Returns

0 on success; -1 if PPU busy

◆ ppu_update()

```
int ppu_update ( void  )
```

Request for the current frame changes to be send to the PPU on the next available frame.

Any previous calls to ppu_set_[...] functions are guaranteed to take effect after this function returns successfully.

If you want to ensure your frame gets sent out to the PPU, and also want to synchronize to the PPU's internal 60FPS timing, keep polling this function until 0 (success) is returned.

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Returns

0 on success; -1 if PPU busy

◆ ppu_write_palette()

```
int ppu_write_palette ( palette_t *  palette,
                        layer_e      layer_id,
                        unsigned     palette_id
                      )
```

Overwrites a palette in Palette RAM.

Overwrites a single `palette` in Palette RAM at `palette_id` and `layer_id`

Recall that Palette RAM is organized into three segments. `layer_id` selects one of these layers: 0: Background Palettes. 16 Palettes 1: Foreground Palettes. 16 Palettes (Else): Sprite Palettes. 32 Palettes

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

palette	A pointer to a palette data struct that we should write to Palette RAM.
layer_id	The target section of Palette RAM to start copying the `palette` to.
palette_id	The id of the palette to overwrite within the given layer. This must be within bounds of the palette layer section indicated by `layer_id`.

Returns

0 on success; -1 if PPU busy

◆ ppu_write_pattern()

```
int ppu_write_pattern ( pattern_t *      pattern,
                        unsigned         width,
                        unsigned         height,
                        pattern_addr_t   pattern_addr
                      )
```

Writes **pattern_t** patterns (8x8-pixel tiles) to a specified location in Pattern RAM.

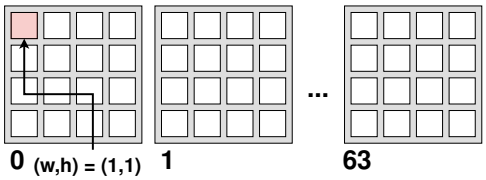
Note

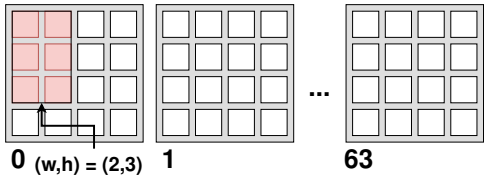
As a reminder, pattern_addr points to a tile chunk and also a specific tile at (x_i, y_i) within.

Within the current tile chunk pointed to by pattern_addr, a subset of `width` **pattern_t** by `height` **pattern_t** is formed.

pattern_t tile-patterns (8x8-pixel tiles) from `patterns` are written sequentially by rows of `width` until `height` rows have been written.

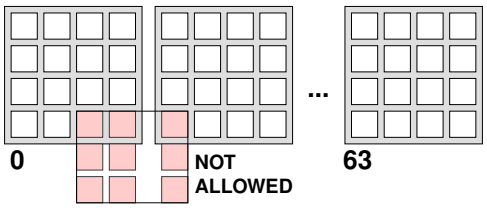
The example diagrams below demonstrate the effect of this function with a variable width/height and pattern_addr of 0:



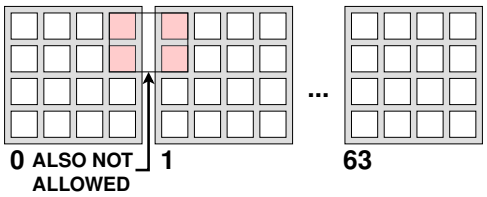


Warning

Note, however that if the subset indicated by width, height, x_i, y_i would extend beyond a 16x16 tile-group boundary, this function will give a warning and abort the program. For example, you are not allowed to write a 3x3-tile pattern starting at (2,3).



Another illegal example for a 2x2-tile pattern starting at (3, 0):



It is the user's responsibility to ensure `patterns` has length `width` * `height` (in terms of **pattern_t**).

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

pattern	The buffer of **pattern_t** tiles. `patterns` must have `width` * `height` total **pattern_t**. Create your buffer so that the **pattern_t** tiles occur row by row sequentially.
width	The width (in 8x8-pixel tiles) of `patterns`. Range [1, 4]
height	The height (in 8x8-pixel tiles) of `patterns`. Range [1, 4]
pattern_addr	The pattern address to start at.

Returns

0 on success; -1 if PPU busy

◆ ppu_write_sprites()

```
int ppu_write_sprites ( sprite_t *  sprites,
                        unsigned    len,
                        unsigned    sprite_id_i
                      )
```

Overwrites one or more sprite data entries in Sprite RAM.

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

sprites	A pointer to an array of sprite data entries to submit to Sprite RAM.
len	Length of `sprites` array.
sprite_id_i	The starting index of the first sprite to overwrite in Sprite RAM. This number must fall in range [0, 63 - `len`].

Returns

0 on success; -1 if PPU busy

◆ ppu_write_tiles_horizontal()

```
int ppu_write_tiles_horizontal ( tile_t *  tiles,
                                 unsigned  len,
                                 unsigned  x_i,
                                 unsigned  y_i,
                                 unsigned  count
                               )
```

Writes an array to a horizontal segment of tiles.

This function copies a buffer of length len tiles into the Tile RAM overwriting count tiles starting at (`x_i`, `y_i`) and moving horizontally. If overwriting count tiles would exceed the boundaries of the logical screen (63, `y_i`), this function will automatically wrap around to the start of the logical screen (0, `y_i`).

If len is lower than count, then this function repeats/tiles the given tiles buffer.

This function is more efficient than ppu_write_tiles_vertical. So if writing a rectangular block of tiles on the screen, prefer to call this function as the inner loop (make the row, `y_i` be the outer loop variable).

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

tiles	Buffer of tile data to write to Tile RAM.
len	Length of the tiles buffer. `len` will be clamped to 64 if len > 64.
x_i	Horizontal position of the first tile to write. Must be in the range [0, 63]
y_i	Vertical position of the first tile to write. Must be in the range [0, 63]
count	The number of tiles to overwrite (horizontally) in Tile RAM. `count` will be set to 64 if count > 64.

Returns

0 on success; -1 if PPU busy

◆ ppu_write_tiles_vertical()

```
int ppu_write_tiles_vertical ( tile_t *  tiles,
                               unsigned  len,
                               unsigned  x_i,
                               unsigned  y_i,
                               unsigned  count
                             )
```

Writes an array to a vertical segment of tiles.

This function copies a buffer of length len tiles into the Tile RAM overwriting coun vertically. If overwriting count tiles would exceed the boundaries of the logical scr automatically wrap around to the start of the logical screen (`x_i`, 0).

If len is lower than count, then this function repeats/tiles the given tiles buffer.

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

tiles	Buffer of tile data to write to Tile RAM.
len	Length of the tiles buffer. `len` will be clamped to 64 if len > 64.
x_i	Horizontal position of the first tile to write. Must be in the range [0, 6
y_i	Vertical position of the first tile to write. Must be in the range [0, 63]
count	The number of tiles to overwrite (vertically) in Tile RAM. `count` will b

Returns

0 on success; -1 if PPU busy

◆ ppu_write_vram()

```
int ppu_write_vram ( const void *  buf,
                     size_t        len,
                     off_t         offset
                   )
```

Write directly to the VRAM buffer.

Attention

This gives a lower-level access to the VRAM buffer! See the higher-level write functions such as the various ppu_write_tiles functions, **ppu_write_sprites**, **ppu_write_pattern**, and **ppu_write_palette**.

Precondition

PPU is currently locked by this process. See **ppu_enable**.

Parameters

buf	Pointer to a buffer to write to the VRAM.
len	Size of buf in bytes.
offset	Byte offset into VRAM.

Returns

0 on success; -1 if PPU busy