nesdoug

My Neslib Notes

Shiru wrote the neslib code, for NES development. These are all my detailed notes on how everything works. I will be adding example code, a little later. I mostly use a slightly modified version of the neslib from

http://shiru.untergrund.net/files/nes/chase.zip (http://shiru.untergrund.net/files/nes/chase.zip)

And, here again is the example code

http://shiru.untergrund.net/files/src/cc65_nes_examples.zip (http://shiru.untergrund.net/files/src/cc65_nes_examples.zip)

And this link has a version of neslib that works with the most recent version of cc65 (as of 2016) version 2.15

http://forums.nesdev.com/viewtopic.php?p=154078#p154078 (http://forums.nesdev.com/viewtopic.php?p=154078#p154078)

pal_all(const char *data);

const unsigned char game_palette[]={…} // define a 32 byte array of chars
pal_all(game_palette);

-pass a pointer to a 32 byte full palette
-it will copy 32 bytes from there to a buffer
-can be done any time, this only updates during v-blank

pal_bg(bg_palette); // 16 bytes only, background

pal_spr(sprite_palette); // 16 bytes only, sprites
-same as pal_all, but 16 bytes
pal_col(unsigned char index,unsigned char color);
-sets only 1 color in any palette, BG or Sprite
-can be done any time, this only updates during v-blank
-index = 0 – 31 (0-15 bg, 16-31 sprite)

#define RED 0x16
pal_col(0, RED); // would set the background color red
pal_col(0, 0x30); // would set the background color white = 0x30

pal_col() might be useful for rotating colors (SMB coins), or blinking a sprite
NOTE: palette buffer is set at 0x1c0-0x1df in example code
PAL_BUF =$01c0, defined somewhere in crt0.s

-this is in the hardware stack. If subroutine calls are more than 16 deep, it will start to overwrite the buffer, possibly causing wrong colors or game crashing

pal_clear(void); // just sets all colors to black, can be done any time

pal_bright(unsigned char bright); // brightens or darkens all the colors
– 0-8, 4 = normal, 3 2 1 darker, 5 6 7 lighter
– 0 is black, 4 is normal, 8 is white
pal_bright(4); // normal

NOTE: pal_bright() must be called at least once during init (and it is, in crt0.s). It sets a pointer to colors that needs to be set for the palette update to work.

Shiru has a fading function in the Chase source code game.c

```
void pal_fade_to(unsigned to)
{
  if(!to) music_stop();
  while(bright!=to)
  {
    delay(4);
    if(bright<to) ++bright;
    else --bright;
    pal_bright(bright);
  }
  if(!bright)
  {
    ppu_off();
    set_vram_update(NULL);
    scroll(0,0);
  }
}
```

pal_spr_bright(unsigned char bright);
-sets sprite brightness only

pal_bg_bright(unsigned char bright); -sets BG brightness , use 0-8, same as pal_bright()

ppu_wait_nmi(void);
-wait for next frame

ppu_wait_frame(void);
-it waits an extra frame every 5 frames, for NTSC TVs
-do not use this, I removed it
-potentially buggy with split screens

ppu_off(void); // turns off screen

ppu_on_all(void); // turns sprites and BG back on

ppu_on_bg(void); // only turns BG on, doesn't affect sprites
ppu_on_spr(void); // only turns sprites on, doesn't affect bg

ppu_mask(unsigned char mask); // sets the 2001 register manually, see nesdev wiki
-could be used to set color emphasis or grayscale modes

ppu_mask(0x1e); // normal, screen on
ppu_mask(0x1f); // grayscale mode, screen on
ppu_mask(0xfe); // screen on, all color emphasis bits set, darkening the screen

ppu_system(void); // returns 0 for PAL, !0 for NTSC

-during init, it does some timed code, and it figures out what kind of TV system is running. This is a way to access that information, if you want to have it programmed differently for each type of TV
-use like…
a = ppu_system();

oam_clear(void); // clears the OAM buffer, making all sprites disappear

OAM_BUF =$0200, defined somewhere in crt0.s

oam_size(unsigned char size); // sets sprite size to 8×8 or 8×16 mode

oam_size(0); // 8×8 mode
oam_size(1); // 8×16 mode

NOTE: at the start of each loop, set sprid to 0
sprid = 0; , then every time you push a sprite to the OAM buffer, it returns the next index value (sprid)

oam_spr(unsigned char x,unsigned char y,unsigned char chrnum,unsigned char attr,unsigned char sprid);
-returns sprid (the current index to the OAM buffer)
-sprid is the number of sprites in the buffer times 4 (4 bytes per sprite)

sprid = oam_spr(1,2,3,0,sprid);
-this will put a sprite at X=1,Y=2, use tile #3, palette #0, and we're using sprid to keep track of the index into the buffer

sprid = oam_spr (1,2,3,0|OAM_FLIP_H,sprid); // the same, but flip the sprite horizontally
sprid = oam_spr (1,2,3,0|OAM_FLIP_V,sprid); // the same, but flip the sprite vertically
sprid = oam_spr (1,2,3,0|OAM_FLIP_H|OAM_FLIP_V,sprid); // the same, but flip the sprite horizontally and vertically
sprid = oam_spr (1,2,3,0|OAM_BEHIND,sprid); // the sprite will be behind the background, but in front of the universal background color (the very first bg palette entry)

oam_meta_spr(unsigned char x,unsigned char y,unsigned char sprid,const unsigned char *data);
-returns sprid (the current index to the OAM buffer)
-sprid is the number of sprites in the buffer times 4 (4 bytes per sprite)

sprid = oam_meta_spr(1,2,sprid, metasprite1) metasprite1[] = …; // definition of the metasprite, array of chars

A metasprite is a collection of sprites
-you can't flip it so easily
-you can make a metasprite with nes screen tool
-it's an array of 4 bytes per tile =
-x offset, y offset, tile, attribute (per tile palette/flip)
-you have to pass a pointer to this data array
-the data set needs to terminate in 128 (0x80)
-during each loop (frame) you will be pushing sprites to the OAM buffer
-they will automatically go to the OAM during v-blank (part of nmi code)

oam_hide_rest(unsigned char sprid);
-pushes the rest of the sprites off screen
-do at the end of each loop

-necessary, if you don't clear the sprites at the beginning of each loop
-if # of sprites on screen is exactly 64, the sprid value would wrap around to 0, and this function would accidentally push all your sprites off screen (passing 0 will push all sprites off screen)
-if for some reason you pass a value not divisible by 4 (like 3), this function would crash the game in an infinite loop
-it might be safer, then, to just use oam_clear() at the start of each loop, and never call oam_hide_rest()

music_play(unsigned char song); // send it a song number, it sets a pointer to the start of the song, will play automatically, updated during v-blank
music_play(0); // plays song #0

music_stop(void); // stops the song, must do music_play() to start again, which will start the beginning of the song

music_pause(unsigned char pause); // pauses a song, and unpauses a song at the point you paused it

music_pause(1); // pause
music_pause(0); // unpause

sfx_play(unsigned char sound,unsigned char channel); // sets a pointer to the start of a sound fx, which will auto-play

sfx_play(0, 0); // plays sound effect #0, priority #0

channel 3 has priority over 2,,,,,, 3 > 2 > 1 > 0. If 2 sound effects conflict, the higher priority will play.

sample_play(unsigned char sample); // play a DMC sound effect

sample_play(0); // play DMC sample #0

pad_poll(unsigned char pad);
-reads a controller
-have to send it a 0 or 1, one for each controller

-do this once per frame
pad1 = pad_poll(0); // reads contoller #1, store in pad1
pad2 = pad_poll(1); // reads contoller #2, store in pad2

pad_trigger(unsigned char pad); // only gets new button presses, not if held

a = pad_trigger(0); // read controller #1, return only if new press this frame
b = pad_trigger(1); // read controller #2, return only if new press this frame

-this actually calls pad_poll(), but returns only new presses, not buttons held

pad_state(unsigned char pad);
-get last poll without polling again
-do pad_poll() first, every frame
-this is so you have a consistent value all frame
-can do this multiple times per frame and will still get the same info

pad1 = pad_state(0); // controller #1, get last poll
pad2 = pad_state(1); // controller #2, get last poll

NOTE: button definitions are opposite of the ones I've used, because they are stored with a shift right rather than shift left

// scrolling //
It is expected that you have 2 int's defined (2 bytes each), ScrollX and ScrollY.
You need to manually keep them from 0 to 0x01ff (0x01df for y, there are only 240 scanlines, not 256)
In example code 9, shiru does this

– -y;

if(y<0) y=240*2-1; // keep Y within the total height of two nametables

scroll(unsigned int x,unsigned int y);
-sets the x and y scroll. can do any time, the numbers don't go to the 2005 registers till next v-blank
-the upper bit changes the base nametable, register 2000 (during the next v-blank)
-assuming you have mirroring set correctly, it will scroll into the next nametable.

scroll(scroll_X,scroll_Y);

split(unsigned int x,unsigned int y);
-waits for sprite zero hit, then changes the x scroll
-will only work if you have a sprite currently in the OAM at the zero position, and it's somewhere on-screen with a non-transparent portion overlapping the non-transparent portion of a BG tile.

-i'm not sure why it asks for y, since it doesn't change the y scroll
-it's actually very hard to do a mid-screen y scroll change, so this is probably for the best
-warning: all CPU time between the function call and the actual split point will be wasted!
-don't use ppu_wait_frame() with this, you might have glitches

Tile banks

-there are 2 sets of 256 tiles loaded to the ppu, ppu addresses 0-0x1fff
-sprites and bg can freely choose which tileset to use, or even both use the same set

bank_spr(unsigned char n); // which set of tiles for sprites

bank_spr(0); // use the first set of tiles
bank_spr(1); // use the second set of tiles

bank_bg(unsigned char n); // which set of tiles for background

bank_bg(0); // use the first set of tiles
bank_bg(1); // use the second set of tiles

rand8(void); // get a random number 0-255
a = rand8(); // a is char

rand16(void); // get a random number 0-65535
a = rand16(); // a is int

set_rand(unsigned int seed); // send an int (2 bytes) to seed the rng

-note, crt0 init code auto sets the seed to 0xfdfd
-you might want to use another seeding method, if randomness is important, like checking FRAME_CNT1 at the time of START pressed on title screen

set_vram_update(unsigned char *buf);
-sets a pointer to an array (a VRAM update buffer, somewhere in the RAM)
-when rendering is ON, this is how BG updates are made

usage…
set_vram_update(Some_ROM_Array); // sets a pointer to the data in ROM

(or)

memcpy(update_list,updateListData,sizeof(updateListData));
– copies data from ROM to a buffer, the buffer is called 'update_list'
set_vram_update(update_list); // sets a pointer, and a flag to auto-update during the next v-blank

also…
set_vram_update(NULL);
-to disable updates, call this function with NULL pointer

The vram buffer should be filled like this…

Non-sequential:
-non-sequential means it will set a PPU address, then write 1 byte
-MSB, LSB, 1 byte data, repeat
-sequence terminated in 0xff (NT_UPD_EOF)

MSB = high byte of PPU address
LSB = low byte of PPU address

Sequential:
-sequential means it will set a PPU address, then write more than 1 byte to the ppu
-left to right (or) top to bottom
-MSB|NT_UPD_HORZ, LSB, # of bytes, a list of the bytes, repeat
or
-MSB|NT_UPD_VERT, LSB, # of bytes, a list of the bytes, repeat
-NT_UPD_HORZ, means it will write left to right, wrapping around to the next line
-NT_UPD_VERT, means is will write top to bottom, but a new address needs to be set after it reaches the bottom of the screen, as it will never wrap to the next column over
-sequence terminated in 0xff (NT_UPD_EOF)

#define NT_UPD_HORZ 0x40 = sequential
#define NT_UPD_VERT 0x80 = sequential
#define NT_UPD_EOF 0xff

Example of 4 sequential writes, left to right, starting at screen position x=1,y=2
tile #'s are 5,6,7,8
{
MSB(NTADR_A(1,2))|NT_UPD_HORZ,
LSB(NTADR_A(1,2)),
4, // 4 writes
5,6,7,8, // tile #'s
NT_UPD_EOF
};

Interestingly, it will continually write the same data, every v-blank, unless you send a NULL pointer like this…
set_vram_update(NULL);
…though, it may not make much difference.
The data set (aka vram buffer) must not be > 256 bytes, including the ff at the end of the data, and should not push more than…I don't know, maybe * bytes of data to the ppu, since this happens during v-blank and not during rendering off, time is very very limited.

> ** Max v-ram changes per frame, with rendering on, before BAD THINGS start to happen…*
>
> *sequential max = 97 (no palette change this frame),*
> *74 (w palette change this frame)*
>
> *non-sequential max = 40 (no palette change this frame),*
> *31 (w palette change this frame)*
>
> *the buffer only needs to be…*
> *3 * 40 + 1 = 121 bytes in size*
> *…as it can't push more bytes than that, during v-blank.*
>
> *(this hasn't been tested on hardware, only FCEUX)*

// all following vram functions only work when display is disabled

vram_adr(unsigned int adr);
-sets a PPU address
(sets a start point in the background for writing tiles)
vram_adr(NAMETABLE_A); // start at the top left of the screen
vram_adr(NTADR_A(x,y));
vram_adr(NTADR_A(5,6)); // sets a start position x=5,y=6

vram_put(unsigned char n); // puts 1 byte there
-use vram_adr(); first
vram_put(6); // push tile # 6 to screen

vram_fill(unsigned char n,unsigned int len); // repeat same tile * LEN
-use vram_adr(); first
-might have to use vram_inc(); first (see below)
vram_fill(1, 0x200); // tile # 1 pushed 512 times

vram_inc(unsigned char n); // mode of ppu
vram_inc(0); // data gets pushed into vram left to right (wraping to next line)
vram_inc(1); // data gets pushed into vram top to bottom (only works for 1 column (30 bytes), then you have to set another address).
-do this BEFORE writing to the screen, if you need to change directions

vram_read(unsigned char *dst,unsigned int size);
-reads a byte from vram
-use vram_adr(); first
-dst is where in RAM you will be storing this data from the ppu, size is how many bytes

vram_read(0x300, 2); // read 2 bytes from vram, write to RAM 0x300

NOTE, don't read from the palette, just use the palette buffer at 0x1c0

vram_write(unsigned char *src,unsigned int size);
-write some bytes to the vram
-use vram_adr(); first
-src is a pointer to the data you are writing to the ppu
-size is how many bytes to write

vram_write(0x300, 2); // write 2 bytes to vram, from RAM 0x300
vram_write(TEXT,sizeof(TEXT)); // TEXT[] is an array of bytes to write to vram.
(For some reason this gave me an error, passing just an array name, had to cast to char * pointer)
vram_write((unsigned char*)TEXT,sizeof(TEXT));

vram_unrle(const unsigned char *data);
-pass it a pointer to the RLE data, and it will push it all to the PPU.
-this unpacks compressed data to the vram
-this is what you should actually use…this is what NES screen tool outputs best.
vram_unrle(titleRLE);

usage:
-first, disable rendering, ppu_off();
-set vram_inc(0) and vram_adr()

-wait for start of frame, with ppu_wait_nmi();
-call vram_unrle();
-then turn rendering back on, ppu_on_all()
-only load 1 nametable worth of data, per frame

NOTE:
-nmi is turned on in init, and never comes off

memcpy(void *dst,void *src,unsigned int len);
-moves data from one place to another…usually from ROM to RAM

memcpy(update_list,updateListData,sizeof(updateListData));

memfill(void *dst,unsigned char value,unsigned int len);
-fill memory with a value

memfill(0x200, 0, 0x100);
-to fill 0x200-0x2ff with zero…that is 0x100 bytes worth of filling

delay(unsigned char frames); // waits a # of frames

delay(5); // wait 5 frames

TECHNICAL NOTES, ON ASM BITS IN NESLIB.S:
-vram (besides the palette) is only updated if VRAM_UPDATE + NAME_UPD_ENABLE are set…
-ppu_wait_frame (or) ppu_wait_nmi, sets ‘UPDATE’
-set_vram_update, sets ‘ENABLE’
-set_vram_update(0); disables the vram ‘UPDATE’
-I guess you can’t set a pointer to the zero page address 0x0000, or it will never update.
-music only plays if FT_SONG_SPEED is set, play sets it, stop resets it, pause sets it to negative (ORA #$80), unpause clears that bit

April 13, 2017April 20, 2017 dougfraker