nesdoug

13. Sprite 0 Trick / Debugging

What is a Sprite Zero Hit? This is one way to time a mid-frame event, such as changing the H Scroll position. That way we can have the top of the screen with lots of text that doesn't move, and have the lower portion of the screen scrolling right and left.

There are several ways to time this.

1.Sprite Zero Hit

2.Sprite Overflow Hit (don't do this)

3.DMC music channel generated IRQ (don't do this either)

4.Some mappers have scanline counters (MMC3)

I'm not going to cover the other ones. The most common one (I think) is the Sprite Zero Hit.

Sprite Zero refers to the very first Sprite in the OAM (addresses 0-3). If that Sprite has non-transparent bits on top of a non-transparent bit of the BG, it sends a signal to $2002 register (bit 0x40). We can use this to time when we change the H scroll. I wrote the Sprite Zero Evaluation bit in ASM.

You absolutely need to make sure the Sprite Zero is always going to be over a non-transparent bg tile, or this will become an infinite loop (crashed game).

First, we do everything that needs to be done in V-blank. Then we set the H scroll to zero (and set the Nametable with a write to $2000 "PPU_CTRL". Then we call the SpriteZero() function, and it will hang out there until it detects a Sprite Zero hit (once the scanline reaches that point of the screen rendering). Then we can switch the H scroll and Nametable.

```
// in NMI code, it sets the H scroll and Nametable to zero, for the top of the screen
Sprite_Zero(); // wait till sprite zero hit
SCROLL = Horiz_scroll;
SCROLL = 0;  // setting the new scroll position
PPU_CTRL = (0x94 + Nametable);
```

And I decided to make the actual Sprite Zero the number zero, to be cute. And I added this bit, which makes it 'disappear' if you press 'Start'. Wow, magic…

```
if ((joypad1 & START) > 0){
SPRITE_ZERO[1] = 0xff; // switch tiles to a very small one
SPRITE_ZERO[2] = 0x20; // attributes = behind the bg
}     // makes it disappear
```

Actually, it's still there, it's just very small, and it's set to be BEHIND the background now. Cool.

———————————-

At this point, I thought this was too simple of a lesson, so I got way over excited and made this into a 4 screen wide scrolling demo, with dynamically generated background that uses a metatile system rather than an RLE compressed background. Here's the rundown…

First, it figures where you are (based on Hscroll position and Room number), and as you scroll right, it will draw 2 new columns of new screen on the Opposite Nametable that you are currently seeing. It does this every 16 pixels of H scroll. And it does it very quick during V-blank. In order to make it as quick as possible, I wrote the PPUupdate (drawing to screen) as a big unrolled ASM loop.

It also reconfigures the BG attribute table, on the fly.

All of this was very complicated, and actually, I had a hell of a time getting it to work correctly…so I'm not going to get into that at all, and instead, I'm going to make this a lesson on DEBUGGING. How to identify and fix errors.

First of all, this scrolling engine is using too many cycles just to complete the basic scrolling functions. How do I know this? At the very end of my infinite loop (in main())…I added this line.

PPU_MASK = 0x1f;

Which turns the screen gray at the point which it's finished with its logic, and waiting for the next V-blank. (You have to have FCEUX set to 'old PPU' to see the gray screen….Config/PPU/old PPU). This is what I was seeing…

Half of my logic time is already gone, and I haven't even added music or enemies. So, I had to figure out where the problem was. For me, the easiest way to do that is to surround a function with a dummy write to an unused bit of the RAM (I usually use 0xff), like this…[TEST = *((unsigned char*)0xff)]

```
++TEST; // for debugging
Should_We_Buffer(); //4422 cycles
++TEST; // for debugging
```

Then, in FCEUX, I open the debugger tool, and set a breakpoint for writes to 0xff. Hit 'run' once, it will break at the first ++TEST. Hit 'run' again, it will break at the next. Look down to 'CPU cycles' and in parentheses, it will tell you how many cycles have passed since the last breakpoint. I tested a few functions, to see which took the longest and the buffering was clearly where the problem was.

I decided to split up the buffering into 2 parts, so they don't both happen on the same frame. Afterward, my screen looks like this…

Much better.

Now, I disabled scrolling left, to make it simpler, and I wanted my guy to be able to run to the left side of the screen (changing his sprite X coordinates). But, for some reason he wouldn't go left. I looked at the code, and it looked fine to me, so I had to do some more debugging.

In the compile.bat file, I added this… -Ln "labels.txt"

Also, you have to put -g on the command lines for cc65/ca65 in the compile.bat file.

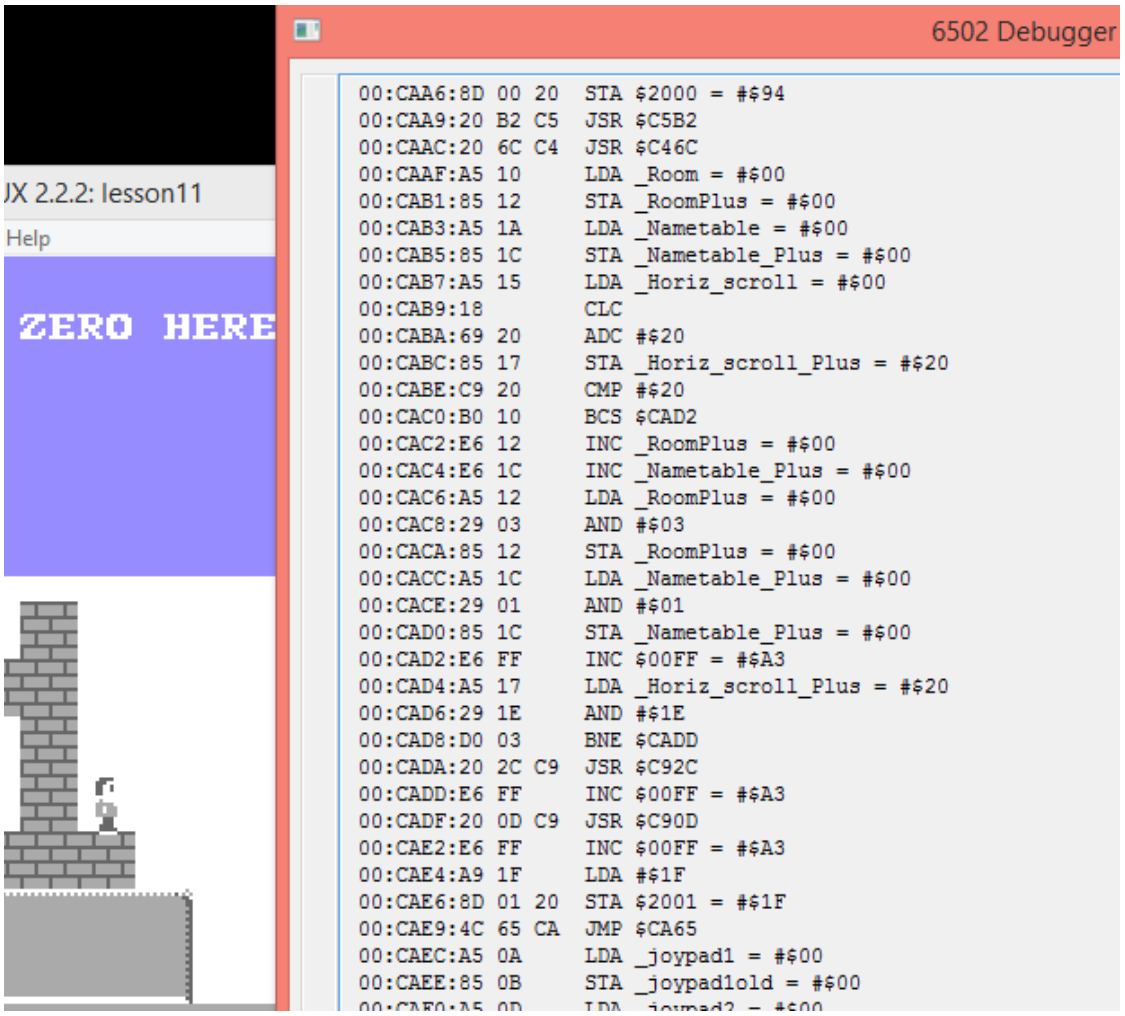
to make the linker generate a list of addresses for all the labels. I was looking for move_logic(). Which, the labels.txt tells me is at C5B2. So I set a breakpoint for execution at C5B2.

(I actually went and made a label that I can stop exactly at the left movement code. The compiler was 'optimizing' out any dummy labels that I made, so in the end I made a dummy function which I called exactly before the code I wanted to review, and we can see in labels.txt that it did work, and I can set a breakpoint for C5AF where my TestLabel is. You don't need to do this, btw. I was just annoyed that it wouldn't let me make an 'unused label' in the c code. Maybe I should have just put another ++TEST instead.)

Unfortunately, all this is in ASM. But I did figure out from "stepping into" the code that X_speed (RAM address 1E) is being reset to zero, even with Left being pressed. Then I discovered that 'if (X_speed < 0)' should have been 'if (X_speed <= 0)'. Aha!

[By the way, in order to have L pressed while breakpoints are on (in FCEUX), I had to set 'auto-hold' under Config/Input/Auto-hold …is mapped to a key on the keyboard, and I press L and the auto-hold key, and THEN set the breakpoint in the debugger.]

Edit: Rainwarrior, over at nesdev, made a python 3 script to convert ca65 label files into FCEUX debugging label file. I've modified it slightly. You need to make a 'labels.txt' file, as described above. Then, run the fceux_symbols3.py file, and give it the name of the game.nes file you're working on. It will generate 3 .nl files. Now, open the .nes game with FCEUX and open the debugger. Now, all of your C variables and labels will be readable.



Here's the link to the python 3 script…(Thanks to rainwarior for permission to use / distribute it.)

http://dl.dropboxusercontent.com/s/fzu98ygo8land19/fceux_symbols3.py?dl=1 (http://dl.dropboxusercontent.com/s/fzu98ygo8land19/fceux_symbols3.py?dl=1)

And, here's the nesdev page with the original version, before I edited it…

http://forums.nesdev.com/viewtopic.php?f=10&t=11114 (http://forums.nesdev.com/viewtopic.php?f=10&t=11114)

-Another edit…I made another version that takes the input as a command line argument, so that it can be added to the compile.bat, and I don't have to run it seperately…

http://dl.dropboxusercontent.com/s/0v8aos6rt6kfghi/fceux_symbols4.py?dl=1 (http://dl.dropboxusercontent.com/s/0v8aos6rt6kfghi/fceux_symbols4.py?dl=1)

…and I added this line to the compile.bat, near the end…

fceux_symbols4.py %name%

Anyway, the 4 screen scrolling engine is working, but it's a bit more complicated than I wanted it to be. You can see the source code, in these links…the second one 'B' is the same thing, but with all the debugging features removed.

I recommend you open the Nametable Viewer and watch what happens while you scroll through the 4 screens. And, again, press 'Start' will make the Sprite Zero 'disappear'. 🙂

http://dl.dropboxusercontent.com/s/08oibyhciz6woi7/lesson11.zip (http://dl.dropboxusercontent.com/s/08oibyhciz6woi7/lesson11.zip)

Note: if you're having trouble getting the linker to generate a label file with ALL your labels, you need to add this line to every .s (ASM) file…

.debuginfo

Or, you need to add the command line instruction -g every time cc65/ca65 is run.

December 2, 2015April 15, 2017 dougfraker