[Mirrors]

Super Nintendo Entertainment System: pinouts & protocol

Contents:

[Document Version: **1.04**] [Last Updated: **22-May-08**]

---

1. About the Author

Author: Jim Christy
Version: 1.02
E-Mail: jchristy@hplred.HP.COM

---

2. Introduction

For all you game hardware enthusiasts out there, I took the opportunity this weekend to put a scope on my Super Nintendo connectors and find out what is going on. Because the standard Multi-out cable connector only has internal contacts for the audio and video signals, I had to find some more push-in gold contacts at a local store to fully break out all the signals. It appears easier to do this than make your own connector.

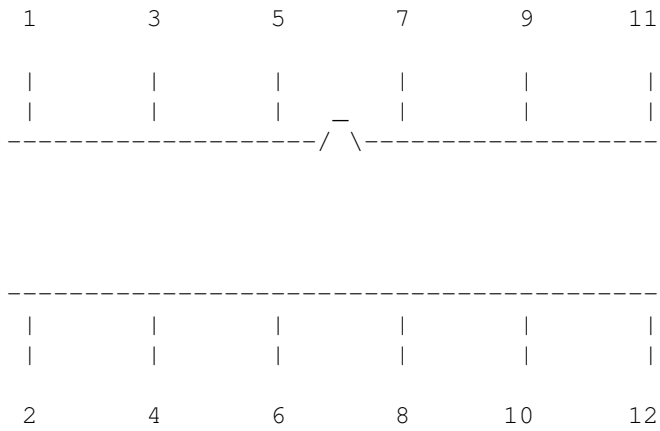
In short, I found that in addition to S-VHS, the multiout also supports RGB and sync. I also got the controller pinouts and protocol, which opens up some interesting possibilities. One could rather easily construct a "macro recorder" that records your exact button presses for a game sequence and allows you to play them back. They will be time-accurate by definition of the protocol, and depending on how random the game plays, you should be able to replay those sequences that get boring, and then take over control when you want.

If all of this is already well known, then sorry for the waste of net bandwidth...

---

3. SNES Multi-out cable connector pins.

These are numbered the way Nintendo did, and the view is looking back "into" the connector on the CABLE.



Pin	Description
1	Red analog video out (1v DC offset, 1vpp video into 75 ohms)
2	Green analog video out (1v DC offset, 1vpp video into 75 ohms)
3	Composite H/V sync out (1vpp into 75 ohms)
4	Blue analog video out (1v DC offset, 1vpp video into 75 ohms)
5	Ground
6	Ground
7	Y (luminance) signal for S-VHS (1vpp into 75 ohms)
8	C (chroma) signal for S-VHS (1vpp into 75 ohms)
9	NTSC composite video signal (1vpp into 75 ohms)
10	+5v (Could be just a high logic signal)
11	Left channel audio out
12	Right channel audio out

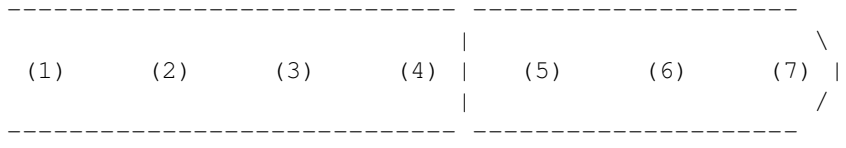
Additional Notes:

As seen above, the SNES does have RGB capability. I was able to get a stable raster on my NEC MultiSync "classic" using the RGB and sync pins. However, the video levels are not RS-170 compatible. The DC offset needs to be filtered out with some large capacitors and the peak-to-peak video amplitude may need to be reduced to 0.7v by using a lower load impedance than 75 ohms. The Y/C (S-VHS) signals *appear* to be directly usable, but tests cannot be made until I find the pinouts for the S-VHS connector on my TV.

---

4. SNES Controller cable connector pins.

I could not find a Nintendo numbering scheme, so I made one up. The view is looking back "into" the connector on the CABLE.



Pin	Description	Color of wire in cable
1	+5v	White
2	Data clock	Yellow
3	Data latch	Orange
4	Serial data	Red
5	?	no wire
6	?	no wire

```
7       Ground                  Brown
```

Additional notes:

Pins 5 and 6 show a DC voltage of 5v on a DMM. I forgot to look at them on a scope so there may pulses too. However, they don't connect to anything at present.

The controllers have a small circuit board with 2 surface mount 14-pin ICs, marked by Nintendo as IC-A and IC-B. Although rubber domes are used to provide the tactile response of the buttons, they are not capacitive technology as originally thought. Instead they use what appears to be carbon impregnated rubber on the underside which makes a resistive path (200 ohms) across 2 carbon coated PCB pads when depressed.

- The red wire goes to pin 2 on IC-A.
- The orange wire goes to pin 8 on both IC-A and IC-B.
- The yellow wire goes to pin 9 on both IC-A and IC-B.

IC-A and IC-B appear to be identical, with a 91 date code and have another (possible part number) of 545. These are most likely 2 parallel load shift registers in series. Buttons on the controller pull the parallel load inputs to ground through the contact formed by pressing a button. IC-B serially feeds IC-A, which then drives the serial data line to the SNES CPU.

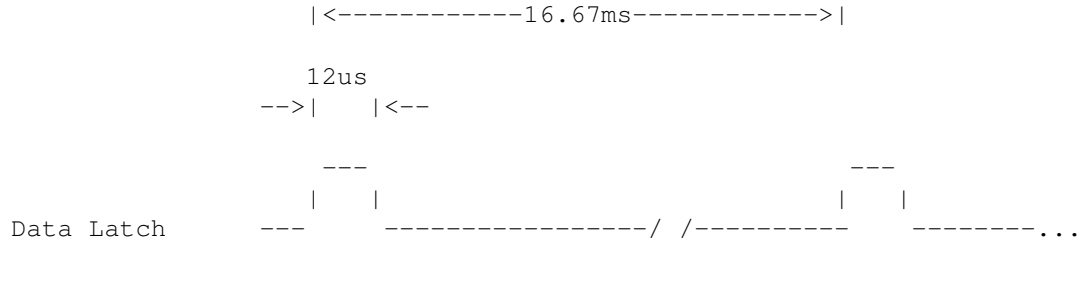
---

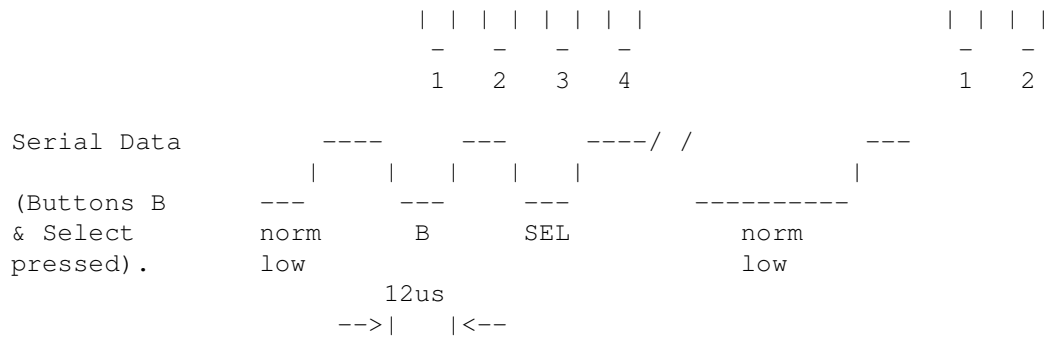
5. SNES Controller Communication Protocol

Every 16.67ms (or about 60Hz), the SNES CPU sends out a 12us wide, positive going data latch pulse on pin 3. This instructs the ICs in the controller to latch the state of all buttons internally. Six microsenconds after the fall of the data latch pulse, the CPU sends out 16 data clock pulses on pin 2. These are 50% duty cycle with 12us per full cycle. The controllers serially shift the latched button states out pin 4 on every rising edge of the clock, and the CPU samples the data on every falling edge.

Each button on the controller is assigned a specific id which corresponds to the clock cycle during which that button's state will be reported. The table in section 4.0 lists the ids for all buttons. Note that multiple buttons may be depressed at any given moment. Also note that a logic "high" on the serial data line means the button is NOT depressed.

At the end of the 16 cycle sequence, the serial data line is driven low until the next data latch pulse. The only slight deviation from this protocol is apparent in the first clock cycle. Because the clock is normally high, the first transition it makes after latch signal is a high-to-low transition. Since data for the first button (B in this case) will be latched on this transition, it's data must actually be driven earlier. The SNES controllers drive data for the first button at the falling edge of latch. Data for all other buttons is driven at the rising edge of clock. Hopefully the following timing diagram will serve to illustrate this. Only 4 of the 16 clock cycles are shown for brevity.





(From: Declan Williams.)

Upon analysis of the behaviour of the SNES with a vacant controller port, I found that the data line is fact held high via a pullup resistor inside the console.

The reason being, the SNES uses active low data, if the line was to be held low, the SNES CPU would think all the buttons are pressed, including opposite directions on the D-pad.

Also, when trying to view the Clock and Latch pulses on an oscilloscope, with no controller connected, the 'scope will show a constant logic "0". The 5A22 (the CPU that is) is in fact still sending out the Clock/Latch signals. The reason that they do not show on an oscilloscope is because the CPU does NOT drive the line high. The CPU either holds the line low (for "0") or leaves it floating (for "1").

So for a "1" to be detected on these signals, the controller (or mouse) must provide a pullup resistor on these signals.

The above information is EXTREMELY important to anyone planing to interface a serial device to the SNES' controller port.

---

6. SNES Controller Button-to-Clock Pulse Assignment

Clock Cycle	Button Reported
1	B
2	Y
3	Select
4	Start
5	Up on joypad
6	Down on joypad
7	Left on joypad
8	Right on joypad
9	A
10	X
11	L
12	R
13	none (always high)
14	none (always high)
15	none (always high)
16	none (always high)

Additional notes:

Clock cycles 13-16 are left high by controllers. On a mouse, cycle 16 is low.

The way the SNES responds if data is low during these cycles varies from game to game. Donkey Kong contry 1 acts as there is no controller in the port, Super Bases loaded 2 does not care.

(From the Editor)

NOTE: S-VHS is not means to mean Super-VHS. It stands for Super-Video (connector and output)

Additional Info (From Kevin Horton)

OK, the SNES uses the 65816 processor, which is basically a 16-bit version of the 6502. It runs at 3.579545 MHz (color-burst), and has an 8-bit data bus. It can address up to 16MB.

The carts are nothing more than ROM. To tell you how much data is one, take the number of 'MegaBits' and divide by 8 to get megabytes. That's how much data is really in the carts. So, an 8-mbit cart really is only 1 megabyte.

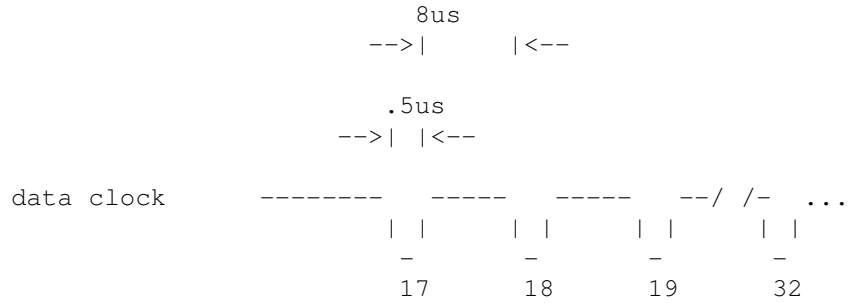
---

7. SNES Mouse Protocol

Supplied by Raphael Assenat

The snes mouse uses the same timing and protocol as a regular pad for it's buttons. The left button is reported at the 9th cycle, and the right button at the 10th cycle. The snes recognizes the mouse when the bit at the 16th clock cycle is low instead of high.

2.5ms after the 16 clock pulses, another series of clock pulses occurs. This is in fact cycles 17 to 32 since no new latch pulse had occured yet. The data is active low, just like the buttons. This time, the clock timing is different:



Here is the meaning of the mouse specific cycles:

Clock Cycle	Button Reported
17	Y direction (0=up, 1=down)
18	Y motion bit 6
19	Y motion bit 5
20	Y motion bit 4
21	Y motion bit 3
22	Y motion bit 2
23	Y motion bit 1
24	Y motion bit 0
25	X direction (0=left, 1=right)
26	X motion bit 6
27	X motion bit 5
28	X motion bit 4
29	X motion bit 3
30	X motion bit 2
31	X motion bit 1

```
32              X motion bit 0
```

Each time the SNES polls the mouse, the mouse reports how it moved since last poll. When no movement occured since last poll, all the motion bits stays high (which means binary 0). The direction bits keep their last state.

Mouse sensitivity:

The mouse has 3 configurable sensitivity levels. The currently active sensitivity level is reported by bits 11 and 12:

- Bit 11 low, Bit 12 high: High sensitivity
- Bit 11 high, Bit 12 low: Medium sensitivity
- Bit 11 high, Bit 12 high: Low sensitivity

Selecting the sensitivity mode:

A special sequence is used to rotate between the 3 modes. First, a normal 12us latch pulse is applied. Next, the first 16 bits are read using normal button timings. Shortly after (about 1ms), 31 short latch pulses (3.4uS) are sent, with the clock going low for 700ns during each latch pulse. For selecting a specific sensitivity, simply execute the special sequence until bits 11 and 12 are as desired.
