Optical Mouse Sensor

Below is a summary of everything I have been able to find about the Logitech g502 optical mouse and mouse sensor, as well as a bit of reasoning on design options moving forward.

The Logitech g502 optical mouse contains a proprietary optical mouse sensor made by Pixart, the Pixart PMW3366DM. This sensor was selected for use in the optical encoder project because it is one of the only mouse sensors available with an inches per second (IPS) rating of over 300. This means that it will be able to keep up with ball throws that may pull line at over 7 m/s. The down side to this mouse selection is that the sensor is currently listed as proprietary, meaning that information about the sensor and the sensor package is very limited at this time.

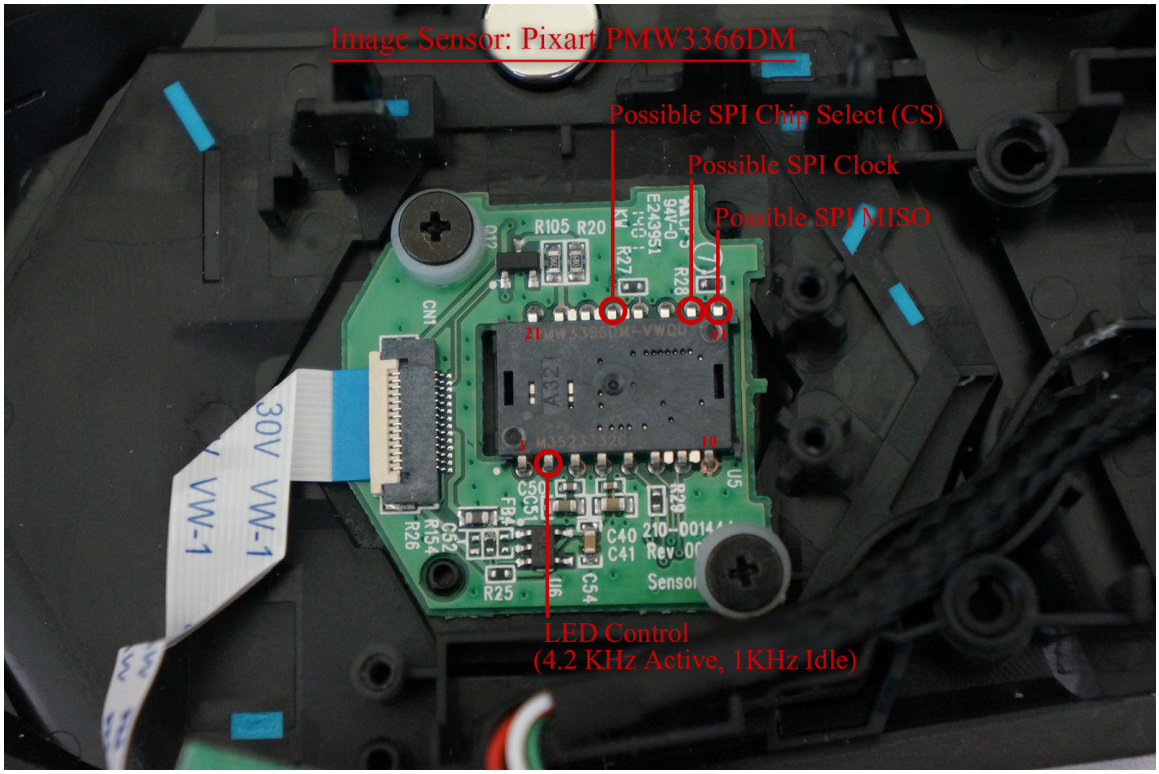
I have written a Linux based program that is capable of using built in plug and play drivers from Linux to disable the mouse as a pointing device and return in tics, the raw data from the mouse in the form of ± a 16 bit number that is the change in position since the last polling. The polling rate should be 1000 Hz. Each tic is dependent on the current resolution setting of the mouse. this can be adjusted from 200 tics per inch to 12,000 tics per inch using Logitech's Gaming software.
(available: http://support.logitech.com/en_us/product/11383#download)

The limitations of this software is its reliance on the Linux operating system in order to set up the USB and Plug and Play environments. This prevents it from being put on a microcontroller or FPGA directly. Possible solutions to this include using a small micro computer, such as a Raspberry Pi, and a lightweight build of Linux to interface to the mouse. This too has problems however, as Linux currently does not have any slave support as it does not natively support external interrupts. This means that while the program that has been written will run on a Raspberry Pi, it is not possible to pole the Raspberry Pi from the xPC target machine via SPI or any other interface. It is for this reason that I then gave up the route of using that program and decided to look more into the sensor itself, to try and get the raw data from the mouse sensor instead of the onboard microprocessor.

What I was able to find by looking at similar Pixart optical sensors that were not proprietary was that many of them use SPI to communicate to the Cortex M processor that is on board. From there I proceeded to disassemble the mouse and, using a portable oscilloscope, probed the optical sensor package to try and determine if I could identify the pinout. From my testing I was able to identify a few pins which I believe with moderate confidence are the data transmission lines of the sensor. I have also identified the pin which controls the pulsing of the IR led of the package, PIN 2. I believe that the data pins are on the right side of the package, on PINS 10 - 21 in the image below. Because I did not have access to a logic analyzer, I made the assumption that the SPI protocol was being used given its use in similar chips. Given that assumption I attempted to identify which PINS corresponded to the Master In Slave Out (MISO), Master Out Slave In (MOSI), Clock (CLK), and Chip Select (CS). I believe that I have identified the

MISO to be on PIN 11 of the sensor package, given that movement data can be seen propagating when the mouse is moved across this line. I also believe that the CLK in on PIN 12 of the sensor package. In the SPI protocol, the CLK is only turned on when there is data being transferred, which matched the behavior seen when probing this line. The chip select has been harder to identify and I am not as confident that I have been successfully able to identify it, however I believe it to be on PIN 16 of the sensor package. I was unable to find anything that I believed to be the MOSI, however there are some SPI device which do not have a MOSI pin as they only have one function. I am not sure if that is the case here.

I would highly suggest using a logic analyzer to try and determine if these results are accurate before moving forward with these findings.

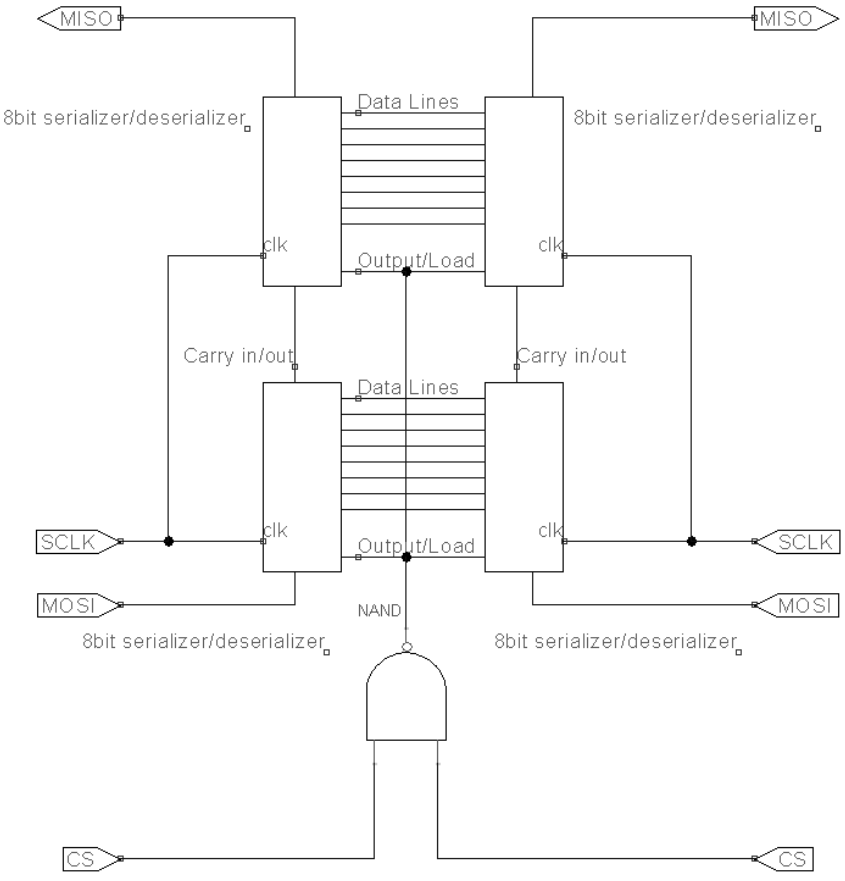


If these PINS are able to be verified to be correct, then one possible option for interfacing with the xPC target machine, is to directly tie into the SPI of the sensor itself. This could be accomplished by severing the connection to the ARM Cortex M directly at the sensor package and then connecting the SPI lines of one of the xPC target machine's FPGAs directly to the sensor package. Then the sensor could be treated as just another SPI device. One thing to note is, if there is a MOSI PIN on the sensor, then a logic analyzer must be used prior to severing its connection, in order to identify what register information it is sending. Once that register is identified, that is the address that must be sent by the xPC target machine in order to be able to pole the device.

Hans Luchsinger

Other Options:

Another option that would remove the need to modify the mouse, would be to build a SPI Master to SPI Master interface. After briefly researching this, I determined that such a device is not available on the market at this time, however I believe that such a device could be built. Such a device would, in effect, act as a buffer between the two master devices. One SPI master would place data into the buffer, and then the other SPI master would read this data. The device would use multiple serial to parallel shift and parallel to serial registers to move and store the data. Data would be serially shifted into the register on the Raspberry Pi side. Then when no select signal or clock signal was high (active), the data would be shifted in parallel to registers on the xPC target machine side. Then, the xPC target machine would read this data in serial again, shifting it out with a known error checking string. This way, when performing a read, it would be able to identify if the data shifted out was the same data that had been shifted in from the previous cycle, indicating that no new data had been received. In this way, the Raspberry Pi and the xPC target machine could be configured to poll the device at a reasonably high rate. While this option sounds like a good one, it still requires additional circuitry to be built and may also suffer from timing issues, specifically dealing with synchronization problems, as well as add additional overhead to the timing of the control loop.



Hans Luchsinger