How system level noise in digital interfaces can lead to spurious errors in serial flash memory

15-03-2020

Authors: Paul Hill, Gordon MacNee

Revision: 1.1

Introduction

In the relentless pursuit of higher performance at a system level, Integrated Device Manufacturers (IDMs) have become well versed in developing digital interfaces that are able to operate at high speeds in electrically challenging environments. Standard interfaces, such as SPI and I2C, provide a relatively simple way of interconnecting devices from different suppliers in a way that is reliable and effective. The same is true for other types of interfaces. The digital domain can be thought of as providing a safe haven for developers looking to build complex systems rapidly using ‘standard’ technologies. Indeed, the embedded industry is largely dependent on standards-based interfaces that ‘just work’, as they provide the framework for innovation. When they don’t ‘just work’, it can lead to confusion, particularly if the cause of a fault is misinterpreted. Any confusion would be understandable, given that the interfaces are developed to be robust and reliable when applied in accordance with the specification. The fact that the underlying physical interface is fixed in silicon would also provide reassurance.

System Noise, in All its Forms

Any distortion to a signal can be interpreted as noise and it is probably reasonable to assume that noise is most often apparent in a communications environment; the signal received is not the signal sent. This direct correlation is relatively simple to find, but in some cases the cause and effect are not so easily identified. The challenge is compounded when the fault becomes intermittent.

Today's microcontrollers are designed to deliver reliable operation with minimal configuration. In the case of a serial interface this may include defaulting to high drive currents on the I/O pins in order to combat the influence of long PCB tracks or high capacitive loads. In some instances, this can result in over-driving an interface which, in turn, can lead to derivative effects that are interpreted as errors or faults.

As an example, serial flash memory devices offer a number of advanced features that ensure reliable operation. This can include noise filters, advanced adaptive programming, and erase algorithms that manage cell margins. Some manufacturers also include ECC in the storage elements saving additional meta data with each write operation to allow single or multi bit errors to be detected and corrected, but this internal ECC fix will not help when noise corrupts the basic message transaction on the communication interface bus.

Noise on the SPI interface can be misinterpreted as additional clock pulses or incorrect data. As SPI is a clock driven interface this would have repercussions such as commands being ignored, data being misinterpreted, wrong commands being used, etc. However, noise also carries energy and in some cases this energy can itself introduce errors in the operation of a device.

Charge Pumps and Overshoot

Some overshoot or undershoot in a signal can be tolerated by digital interfaces. It should not be forgotten, however, that the energy under the curve is still present and this can be disruptive.

A case in point is the charge pump circuitry inside the serial flash memory that generates the voltages to program and erase the memory cells. If the SPI bus signals contain significant noise, there is a chance that the energy in that signal can propagate through to the charge pump and disrupt its operation.

The charge pump in flash memory is a critical function, as it provides the power needed to change the bias of a memory cell and, effectively, store a logical 1 or 0. The write/erase process is a crucial time in the operation of flash memory, any disruption to the charge pump during this time can cause write or erase errors and, while these errors may be detected, there is a chance that they will not be apparent.

An error of this sort can easily be interpreted as a fault in the flash memory device. The fact that flash memory has a finite number of write / erase cycles guaranteed by the manufacturer is well understood by embedded designers, but what is perhaps not so well understood is the importance of providing a clean interface devoid of too much overshoot or undershoot (noise).

As an example, consider the image in Figure 1. It shows healthy cell margins for six flash devices. Two distinct patterns (lobes) emerge between cells programmed with data representing logical 1 (2V to 5V) and 0 (>6v). In Figure 1 there is good separation between the left and right lobes. By comparison, the image in Figure 2 shows memory cell margin for three flash devices that have suffered data corruption caused by noise (overshoot and undershoot) on the control lines. There is no separation between the lobes and the memory cells values for 1's and 0's begin to merge

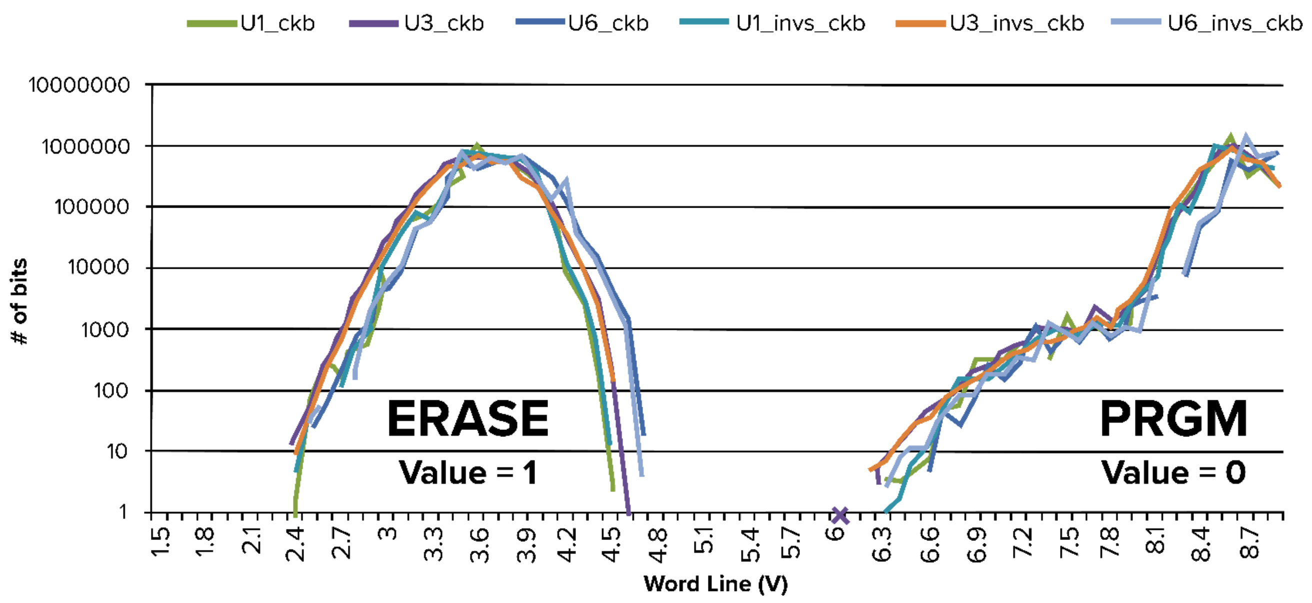


Figure 1: This image shows good cell margin separation data for flash memory that has been programmed and erased.

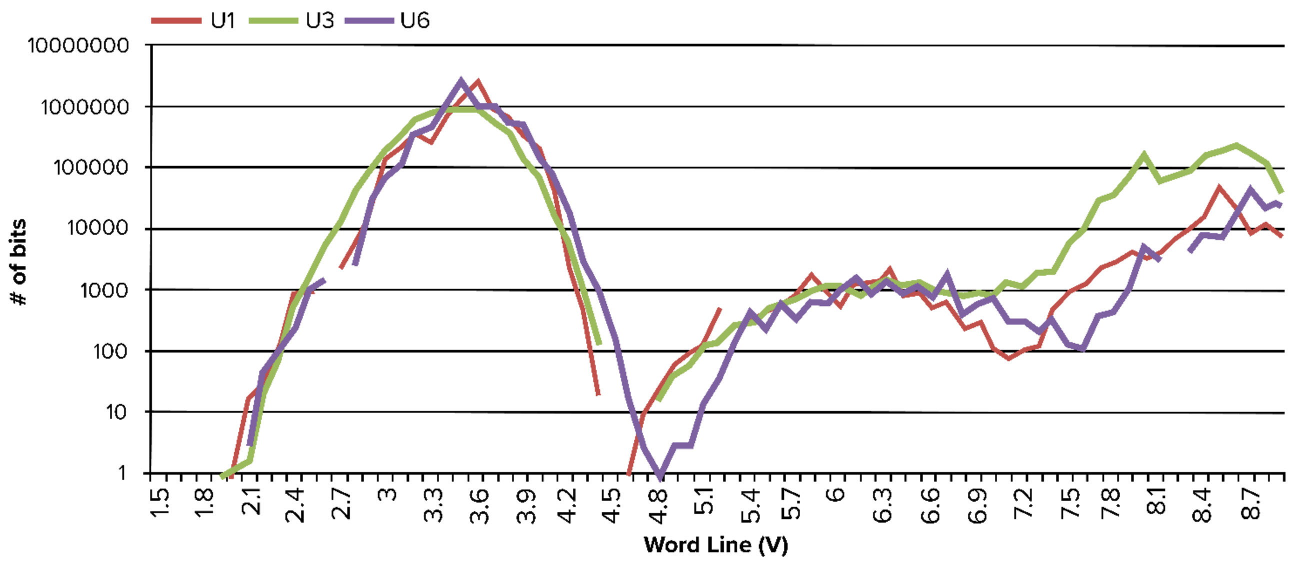


Figure 2: This image shows poor cell margin separation data for flash memory where there has been significant noise present on the SPI lines

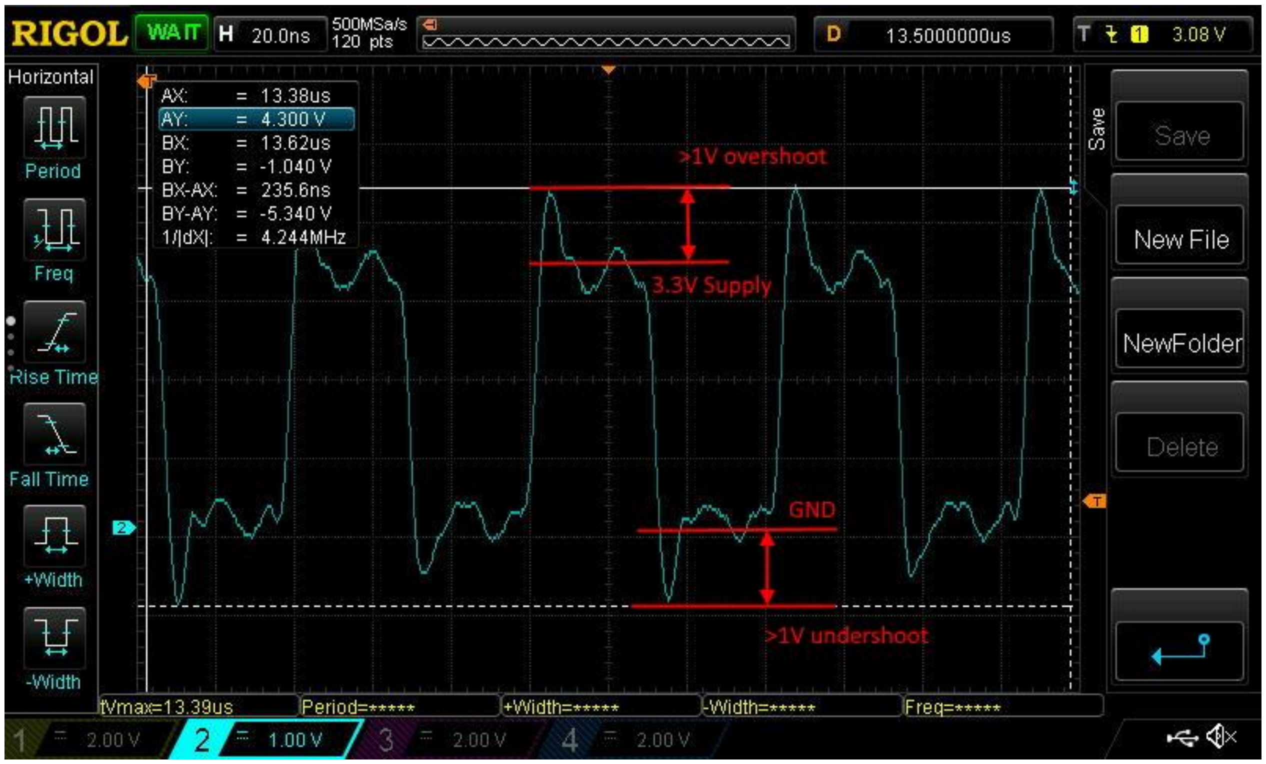
Multiple factors can contribute to the noise level, such as frequency of operation, amplitude of the signal, MCU drive levels, and the energy contained in the noise spikes. PCB design and cross talk between signals.

The data in Figure 2 shows the effects of excessive noise on the serial interface. Figure 3 below shows a representation of what this overshoot would look like in a real application.

The result of this noise was erroneous device operation, which showed up as errors in the serial flash memory stored values. Initially, the true impact of the errors was missed, as polling the STATUS register on a less frequent basis reported fewer errors, leading the designer to make incorrect assumptions on the root cause of the failure.

Although this fault appeared as a memory failure, the root cause was not with the Flash devices. This was discovered by Dialog engineers by probing the SPI signals and identifying possible effects of noise in the system and disruption to the stored memory contents. This was verified by taking a ‘faulty’ device where the memory cell margin analysis was shown in Figure 2 above, and then erasing and re-programming the same device in a clean environment and showing that the cell margins returned to a normal profile map as shown in Figure 1 above.

While the noise could be partly attributed to an impedance mismatch present on the PCB track between the MCU and Flash memory, it wasn’t the full story. The source of the noise was actually proven to be due to the high MCU default I/O Drive levels, which defaults to a highest drive level at power-up. The excessive drive was enough to cause overshoot and under-shoot on the SPI lines, which in some cases can be misinterpreted as signal transitions, leading to read or write errors. However, in this instance it was found that the overshoot held sufficient energy to disrupt the Flash charge pump, which was in turn causing programming and erase errors.


AX: = 13.38us
AY: = 4.300 V
BX: = 13.62us
BY: = -1.040 V
BX-AX: = 235.6ns
BY-AY: = -5.340 V
1/|dX|: = 4.244MHz

tVmax=13.39us Period=***** +Width=***** -Width=***** Freq=*****

Figure 3: This trace image clearly shows the overshoot and undershoot present on the SPI lines were resulting in a peak-to-peak voltage of 5.65V, which exceeds the absolute maximum value documented in the Flash memory specification.

In the customer's design, the microcontroller being used provided a configurable drive current for its I/O, which defaults to HIGH at start-up. As the application code did not modify this level during initialization it remained high in normal operation. The impact of this may not be apparent for other devices on the SPI bus, as digital interfaces are typically designed to be robust. The nature of flash memory, the need to operate at much higher frequencies, and in particular, the operation of the charge pump, made the memory susceptible to large overshoot/undershoot. This led to erroneous operation that was initially misinterpreted as a fault in the flash memory device. Reducing the drive current, through firmware, reduced the overshoot and undershoot to effectively zero (Figure 4), and in turn resulted in error-free operation of the flash memory.

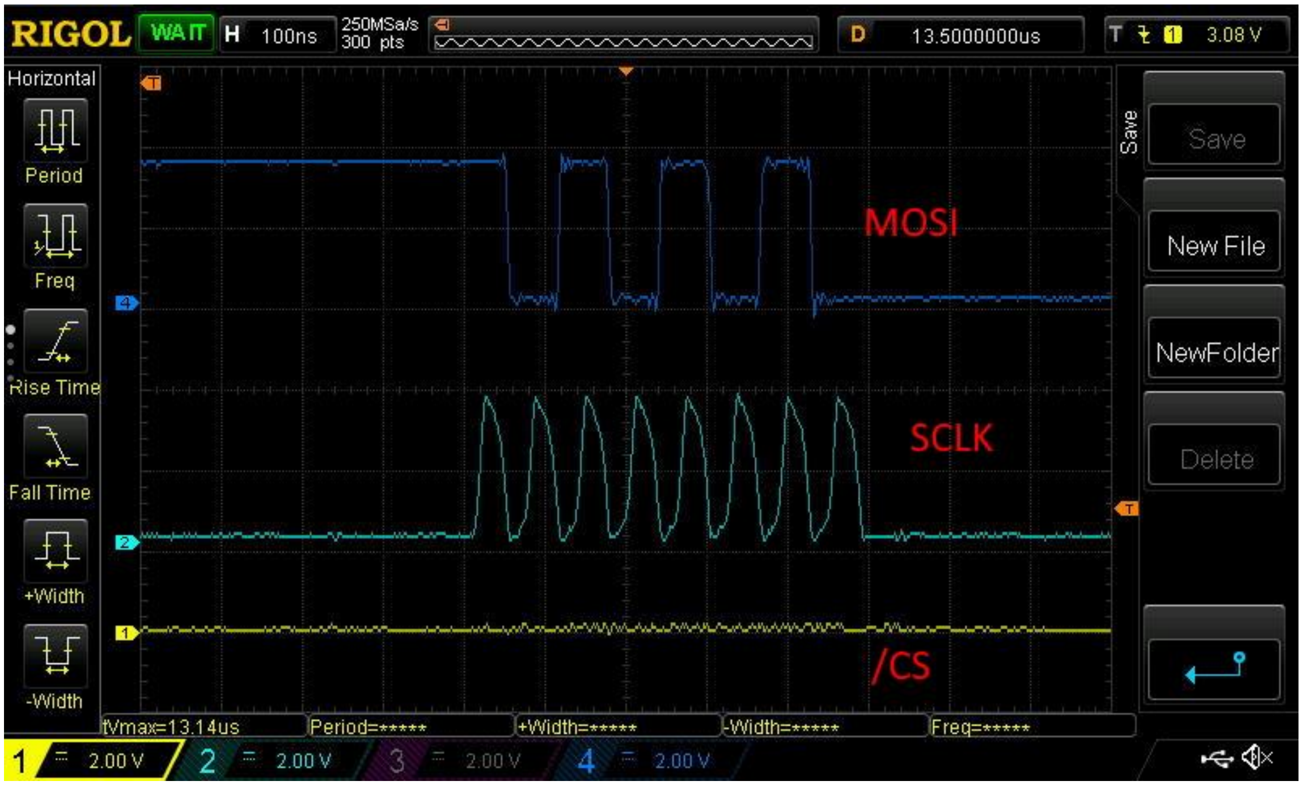


Figure 4: With no overshoot apparent, the serial Flash memory's charge pump was able to operate correctly and provide reliable functionality

The nature of the fault showed that the flash device was making every effort to compensate for the effect of the error, which was the excessive system noise on the SPI interface. Perhaps the most significant point here is that the cause was actually a design feature of the MCU used, which defaulted to an operating mode that, in most situations, would be entirely acceptable. The combination of a high drive output and imperfect PCB inductance created a condition that resulted in intermittent failure. Reducing the drive output on the MCU, through a simple firmware change, solved the issue.

Conclusions

System noise can easily be dismissed when there is no apparent impact. Intermittent errors are particularly difficult to locate under optimal conditions, but when the errors are misinterpreted it makes the challenge even harder.

Overshoot is possibly the least apparent form of system noise, but as explained here, its impact can be significant. Flash memory is a reliable technology but one that depends on a carefully designed interface. Excessive noise on the serial interface has the potential to propagate through to the charge pump circuitry, detracting from the operation of the programming and erase circuitry. This results in unforeseen characteristics that can easily be interpreted as a fault on the device itself showing up as memory cell faults, as well as inconsistent or unreliable programming and erase operations.

In this instance, replacing the flash memory and assuming the issue was solved could have resulted in products going to market that were likely to fail at some point. Instead, the designer was able to improve programming and erase consistency by a significant factor, with effective endurance jumping from an unacceptable ~20K cycles before errors were detected to over 2.5M cycles with no errors (customers own results) and no requirement for supplementary error detection and correction routines.

The level of configurability offered by modern microcontrollers can be seen as both a help and a hindrance; the fact that the drive current is configurable was possibly the cause for the overshoot in this example. However, being able to reduce the drive strength was also effective in solving the problem.

In addition there were other potential benefits not investigated by this article such as subtle improvements in system power consumption, reduced electro-magnetic emissions, easier product certification, and above all, a more reliable product with higher performance - leading to fewer field returns and greater customer satisfaction.