Flash endurance testing

Since flash erase/write cycles are fairly limited, on several recent microcontroller projects I have needed to pack bits carefully to ensure that gadget lifetimes would be reasonable. To increase gadget lifetime, error correction codes are often used to help prevent data corruption, but different error codes have different performance characteristics. Knowing more details about bit failure modes, patterns, and related issues would help select a code that is most useful for my combination of needs and restrictions, but I could not find the information I wanted, so I decided to test my devices. This article details the flash bit failures for the device I tested, and hopefully is useful for similar devices and designers.

To gather the relevant data, I wrote code that repeatedly erased/wrote flash memory, logging all failures so I could then analyze what happened. The results are interesting.

Todo – 1 million cycles, 1 billion errors, 1 day 9 hours

# Background

Since flash degrades over time from erasing/writing (or even just reading, although that decay is slower), if the decay modes are understood, then error correction codes can be designed to improve the length of time before a system fails. However the two main error codes recommended, Reed Solomon and BCH codes, are suited for different failure patterns. Without getting into the details (that may be another article), Reed Solomon is more useful for errors that are in bursts, which BCH codes are better for bit errors that are equidistributed among all bits. (Informally, both work over finite fields of size 2^n, and RS codes correct field elements, i.e., n bits at a time, while BCH codes work more on the bit level).

Another issue was the devices I was using listed erase/cycle lifetime of about 20,000 cycles, and I wanted to test some devices to see how accurate this is.

To help explain the results, it is useful to cover the physical level of flash storage.

## Semiconductors

Since I wanted this to be approachable to non-physicists/non-solid state people, I’ll explain how flash memory works starting at a very low level: atoms and electrons.

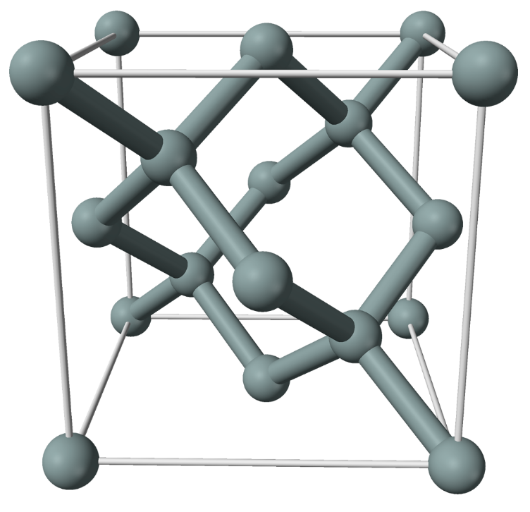
Electrons are most accurately treated by quantum field theories (QFTs), which are a bit too abstract, so I will treat electrons (incorrectly, but illustratively) as little balls with charge that orbit the nucleus of an atom, with the non-classical behavior of being able to quantum tunnel through energy barriers. Electrons have a -1 charge, and each proton in the nucleus has a +1 charge. As described theoretically by QFTs (or less precisely by quantum mechanics), and as determined experimentally, electrons bound in atom orbits have discrete energy levels. Also as described theoretically and determined experimentally, opposite charges attract and like charges repel. These basic physics facts drive most of how solid state devices work (to get an accurate quantitative model takes a lot more quantum mechanics and solid state physics, but this simple level gives a good understanding).

Atoms bond with each other to form larger structures by trying to form lower energy states, which includes trying to fill electron shells and/or taking electrons from atoms that have weaker hold on them (with the resulting + and – charged ions then forming ionic bonds).

Just like a gravity field is a force that attracts mass, an electric field is a force that attracts (or repels) electric charge. The strength of a gravitational field is similar to voltage: a stronger gravitational field causes masses to accelerate more, and a higher voltage causes an electric charge to accelerate more. This is needed to understand how electrons are moved about in semiconductors, and in particular, flash memory.

The ease that materials allow electrons to move about in the presence of an electric field is called conductivity, and the reciprocal is called resistivity. At a given voltage (think gravitational pull for charges), a more conductive material will allow more electrons to flow, and the rate of electron flow (the flux) is called current. Increase the voltage, and the current will increase. All of this is somewhat simplistic, but accurate to first order, and illustrative of how devices work.

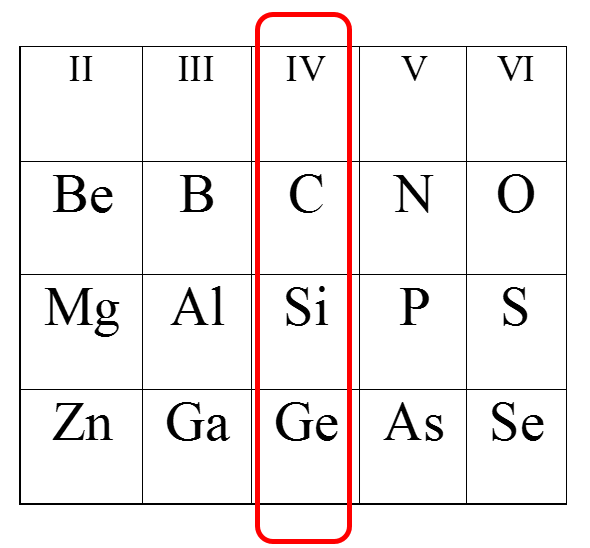
This conductivity is caused by the ability for electrons to wander through the material: if the electrons are bound tightly they are harder to move. Metals have lots of easily moved electrons, making them conduct easily. The amount of electrons and how easily they move is related to the atomic structure in the atoms and how they are arranged.

Elements are organized in the periodic table into columns that give a first approximation to outer electron behavior. Those with 4 outer electrons (Group IV, including silicon, carbon, and germanium) contains what are called “elemental semiconductors,” which are materials with conductivity that lies somewhat between conductors and insulators. In general, a conductor has around 1 to 2 freely moving electrons per atom, semiconductors have 1 electron per 10^7 to 10^17 electrons.

The outer shell for atoms in this (and most parts) of the periodic table try to grab 8 outer electrons (which is why the noble gases, with 8 outer electrons already, do not easily form compounds). Since each in Group IV comes with 4 already, they each want 4 more. Forming a crystal of carbon, silicon, or germanium causes the atoms to form a diamond cubic crystal lattice, in which each atom sits at the center of a tetrahedron, and connects to 4 neighbors. In this form, each atom has shares the 4 outer electrons with 4 neighbors, and each neighbor shares one back. Think of each atom as getting the desired 8 outer electrons part of the time, and sharing part of the time.

The lattice can be viewed as having atoms at each corner and face of a cube, with bonds shared as in the diagram. The electrons forming bonds pull the lattice together and the positive charges in the nucleus repel to hold the spacing apart.

Of these Group IV elements, silicon is used as the basis for modern semiconductors, because of multiple reasons: it forms high quality compounds like the insulator silicon dioxide (SiO2), it has a nice bandgap (long story!) of 1.1 eV, making it perform well at room temperatures, it is cheap, and it is easy to build into low defect atomic structures.

Atoms from neighboring columns have either 3 or 5 outer shell electrons, and carefully mixing them in when making the silicon lattice results in a material with either more or fewer electrons than the eight per outer shell that is energy minimizing. These atoms take the place in the lattice of what would have otherwise been silicon.

Resulting free electrons move much more easily than the bound ones. For example, adding Group V phosphorus (P) adds one free electron per atom which can move much more freely than electrons in pure silicon, increasing conductivity. Adding Group III aluminum (Al) would result in one fewer electron per Al atom, and now instead of an excess electron, there is a shell missing an outer electron, which results in a pull on any electron to try and fill the shell. Conceptually this is called a hole, and treated like a moving positive charge, when in fact it is physically merely a pull on negative charges (electrons) to fill the hole. Boron is the most widely used p-type dopant, phosporos and arsenic are the most widely used n-type dopants.

The result of careful adding of the right amount of the right kinds of atoms to the silicon crystal as it is formed can change the conductivity drastically. Substitution of just one dopant atom into 10^7 atoms of Si can increase the conductivity by a factor of 100,000.

TODO:

Object negatively charged if excess of electrons, positively if not enough electrons to match the number of protons.

Coulomb (C) is quantity of charge, 6.241x10^18 electrons. Ampere (A) is coulombs per second. Resistivity is in ohm \* meters, conductivity is reciprocal, in siemens \* meter

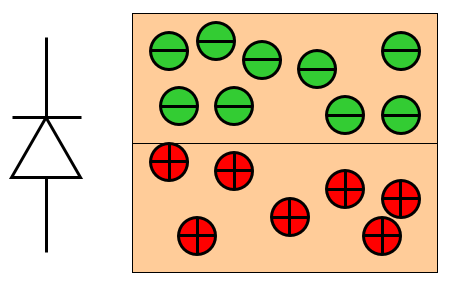
Volt = potential, 1 Volt is the potential difference between two infinite parallel conducting planes 1 meter apart aht creates a field of 1 Newton per Coluumb. 1 Newton is the force rgfavity on 102g, or 1kg forces 9.81 newtons pulling down.

Resistivity = electric field/current, E in Volts/meter, current in Amp/meter^2

TODO show # electrons flowing in gold, silicon, and something doped.

## Diodes

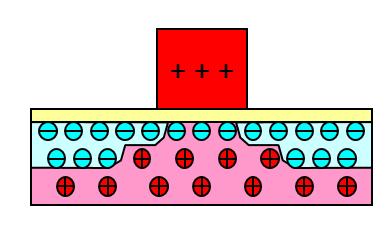
p-n junction: current can only flow one way



TODO

## Transistors

TODO



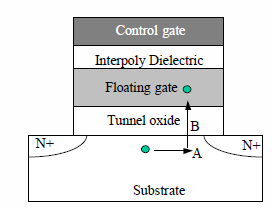
## Flash memory

Now I can explain how flash memory works.

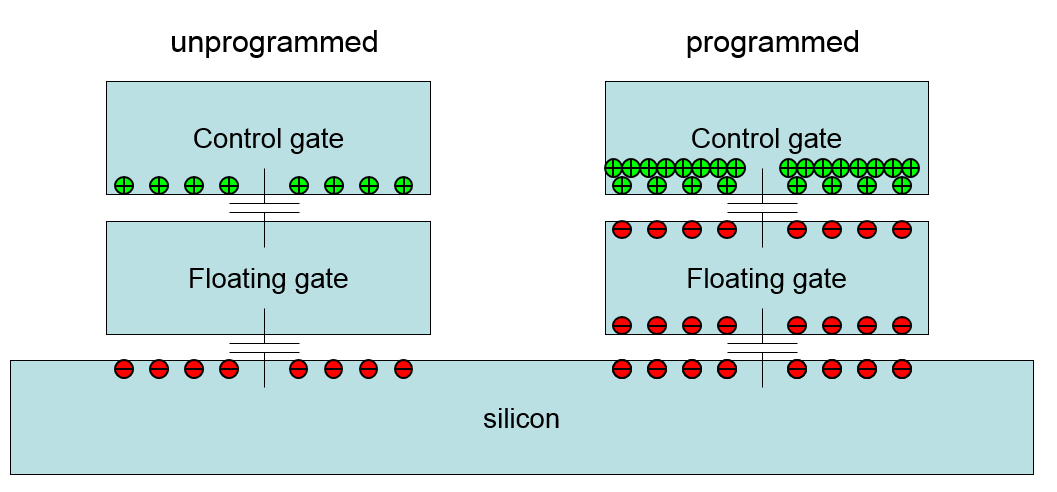
Flash memory is made from transistors of a special construction. Instead of the usual TOD

Flash memory comes in two main types, NAND and NOR (named after the underlying gate), one difference being whether or not the memory allows random read accesses. NAND allows memory to be erased or written in blocks, while NOR allows single cells to be read. Since NOR requires more circuitry, it is more expensive per cell. As a result, most microcontrollers use NOR for storing execute-in-place code, since it is more efficient to execute code from memory allowing random access.

Silicon dioxide used as tunner diletric, lower voltage requires thinner oxide thickness. Cannot go under 7nm if data retention of 10 years needed.

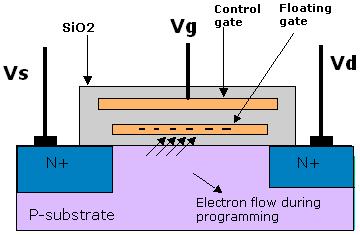
Programmed by hot electron injection (electon energy needed to be near 3eV (lower current = slower programming). Hihg voltage (potential) needed to generate high energy electrons these electrons.

Electron accelerates from source to drain (A), under positive drain bias. (float left, -5V right, 6V control gate)



To obtain same channel charge, programmed gate needs higher control-gate voltage than unprogrammed gate.

TODO – silicon overview, doping, etc.

[](http://www.eeherald.com/)

NAND accessed like block devices, each block consists of pages (512, 2048, and 4096 bytes in size are common). Additional bytes add error correction. Reading and writing are page level, erasure is block level. To execute code form NAND, it is usually copied to RAM first, using some form of memory management unit.

Both erase in pages,

Flash stores information at the physical level by storing groups of electrons in insulated containers, and then a gate on a transistor allows reading whether or not there is charge stored in the container by using the electrostatic pull of those electrons. Multi-level has TODO.

Electrons are “pushed” into the container via hot tunneling (TODO), and removed via quantum tunneling (TODO). Hot tunneling happens when fast moving electrons penetrate the insulating barrier and get trapped in the container. Over time this causes the material sto break down, TODO. Quantum tunneling is accomplished TODO.

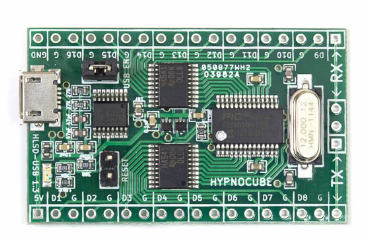
When the memory is erased via quantum tunneling, the memory reports this as a logical ‘1’, and when the container holds enough electrons, the memory reports a logical ‘0’.

Erase cycles slow (because of method?) write cycles fast (todo – speed?)

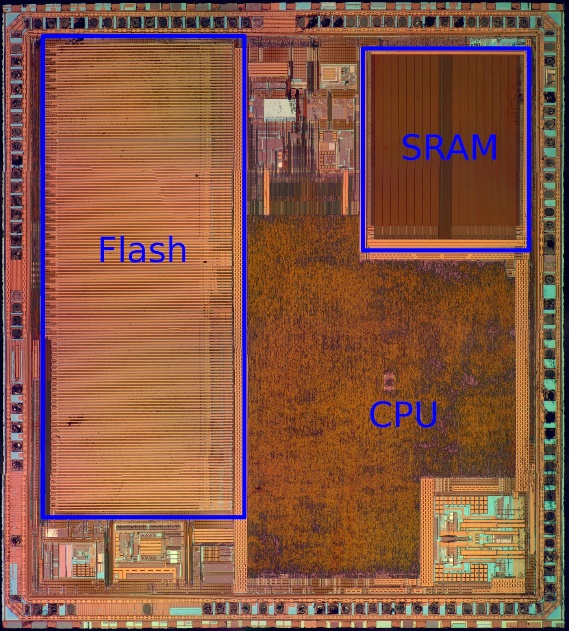
Wikipedia lists common write endurance of

* SLC NAND flash is typically rated at about 100 k cycles (Samsung OneNAND KFW4G16Q2M)
* MLC NAND flash is typically rated at about 5–10 k cycles for medium-capacity applications (Samsung K9G8G08U0M) and 1–3 k cycles for high-capacity applications[citation needed]
* TLC NAND flash is typically rated at about 1 k cycles (Samsung 840)
* SLC floating-gate NOR flash has typical endurance rating of 100 k to 1 M cycles (Numonyx M58BW 100 k; Spansion S29CD016J 1,000 k)
* MLC floating-gate NOR flash has typical endurance rating of 100 k cycles (Numonyx J3 flash)

# The experiment

Now that I have covered the underlying principles, here is the experiment we did on an actual device to see how flash degrades. The microcontroller we used is a PIC32MX150F128B [PIC150], mostly because we already had many nicely hackable boards we developed using this PIC in the form of our HypnoLSD modules (LED strand controllers). This PIC has 32K of RAM, 128K of NOR flash, and we ran it at 48MHZ with a 1Mbaud serial connection to output logging data. We ran 1 million erase/write cycles on a device rated for 20,000 cycles.

This PIC has a flash page size of 1024 bytes, a row size of 128 bytes. Flash is addressable as 32-bit words.

There is 128K of NOR flash and an additional 3K of flash for a boot loader. The hardware has support for 1) erasing all of flash, 2) [](http://4.bp.blogspot.com/-i1D9DFXLcNE/UzCS-BQSweI/AAAAAAAAAgU/yj2pXiJxpBM/s1600/pic32mx340f512h_m4_bf_neo5x-4k_annotated.jpg)erasing one page, 3) programming one row, and 4) programming one word.

From [siliconporn] a related PIC device, the PIC32MX340F512H, has a bit cell pitch of 1015x676 nm (which is 0.686 μm^2/bit), so I suspect the PIC we tested is made similarly. The die of the PIC32MX340F512H is shown; I could not find a die shot of our particular PIC.

I expected once a bit failed it would later work again, then fail again, etc. Thus I only wanted to log changes between failing and succeeding to lessen the amount of messages logged. To track these changes I needed RAM buffers to compare last state and new state for each bit tested in flash, and I needed to track last erased result and last written result, so I needed 2 bytes of RAM for each byte of flash I wanted to test. I chose to test 2 flash pages at a time, so I needed 4096 bytes of RAM.

The code on the PIC is a small C application that sets up the hardware, then loops over erase/write cycles, writing any failures to the output serial port. I logged the output to a file, which took around 24 hours to log 1,000,000 erase/write cycles.

The code for the PIC portion is at <https://github.com/ChrisAtHypnocube/FlashWrecker>.

In slightly more detail, the code loops over the following:

1. Erase pages (which is supposed to set all bits to ‘1’)
2. Read pages checking all bits were set to ‘1’, logging any errors. Erased pages are copied to RAM to only list \*changes\* to states over time.
3. Write zeros to all memory spots.
4. Read pages checking all bits were cleared, logging any errors. Written pages are also copied to RAM to only list \*changes\* to states over time.

The resulting log file is 4.6GB of text which has the form:

**...**

**Pass 723466, frame 0, offset 00000000, time 908b5feb, errors 824483**

**ERROR: (E) offset 0000001E read FFFFFFFB desired FFFFFF7B.**

**ERROR: (E) offset 00000046 read FFFFFFFF desired 7FFFFFFF.**

**ERROR: (E) offset 00000084 read EFFFFFFF desired FFFFFFFF.**

**ERROR: (E) offset 0000008E read FFEFFFFF desired FFFFFFFF.**

**ERROR: (E) offset 000000B7 read FFFFFFDF desired FFFFFFFF.**

**ERROR: (E) offset 000000C4 read FFFBFFFF desired FFFFFFFF.**

**ERROR: (E) offset 000001B8 read FF7FFFFF desired 7F7FFFFF.**

**ERROR: (E) offset 000001BE read 7FFFFFFF desired FFFFFFFF.**

**ERROR: (E) offset 000001D2 read FFFFFF7F desired FFFFFFFF.**

**Pass 723467, frame 0, offset 00000000, time 90aea31f, errors 824492**

**ERROR: (E) offset 00000046 read 7FFFFFFF desired FFFFFFFF.**

**...**

# Analysis

To analyze the data, I made a C#/WPF program (at <https://github.com/ChrisAtHypnocube/FlashWreckAnalyzer>) that reads in the log file, allows interactive visualization of the time changes of the flash, computes some stats, and ultimately creates a visualization of the flash degradation.

A final visualization of the million frame degradation is at TODO

The first bit error occurred on pass TODO. A separate run on a separate device had this happen at cycle TODO. So (given the small sample size of 2!) it appears that 20,000 cycles is quite conservative (I could not find a detailed description of how Microchip arrived at their value; perhaps it was an earlier manufacturing process, and the data sheets are out of date…. Who knows).

The main thing I wanted to test is the accuracy of the statement: “each bit is independently likely to fail, with exponential dependence on number of erase/write cycles”.

Mathematica TODO

Other interesting things I tested

1. Words with n bits failed
2. Not burst errors

GET:

1. frame # where new bit fails for each new failure
2. Bit fails, then works, then fails, data about this?

TODO “The Flash page size on PIC32MX-1XX/2XX devices is 1 KB and the row size is 128 bytes (256 IW and 32 IW, respectively).”

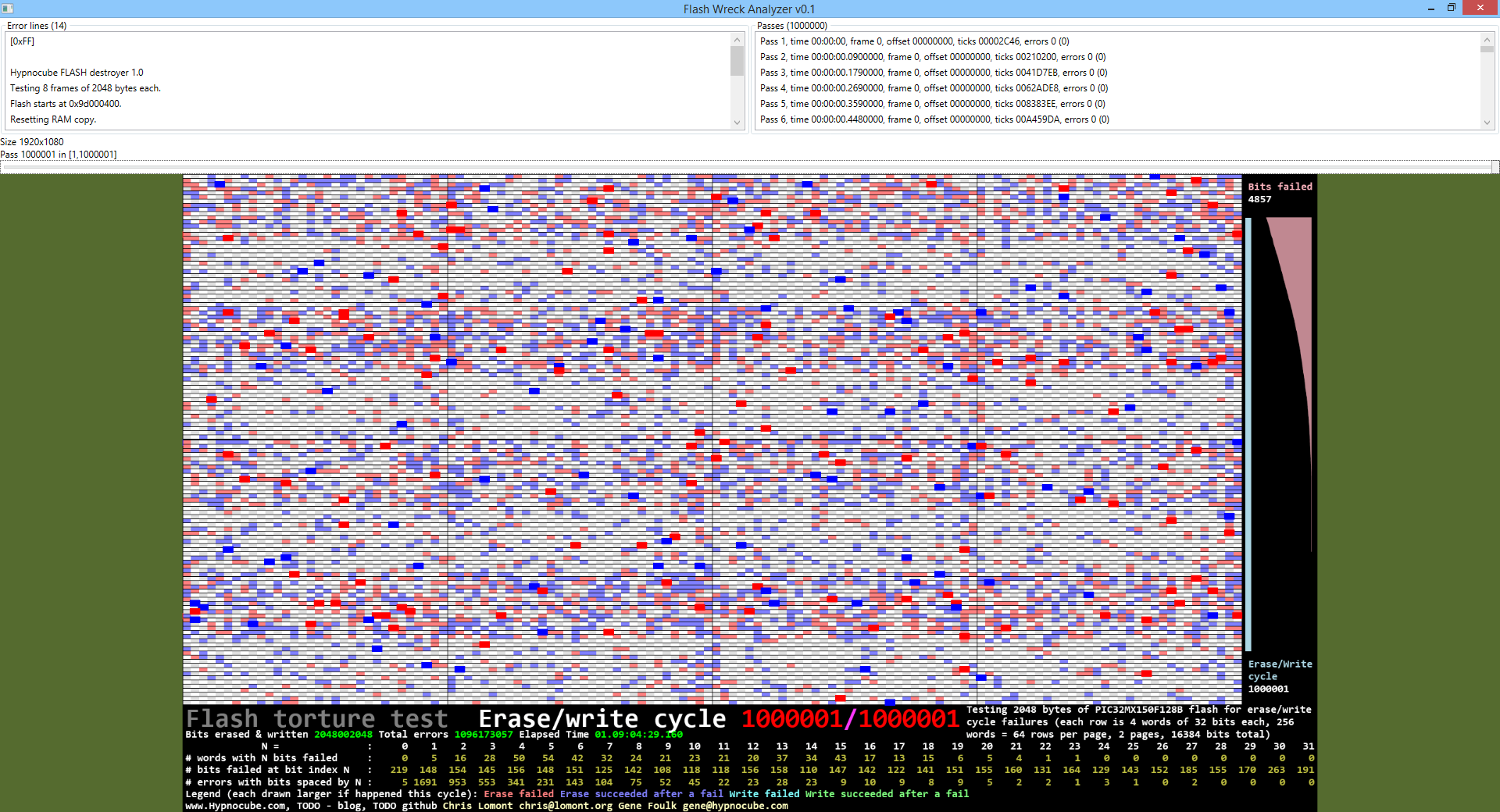
TODO – note all failures were erase, not write, failures

TODO – note we should test read failures later (write once, read flash around an address until something fails)

The microcontroller

TODO – how does our data match bit errors being randomly spread?

After reaching enough erase/write cycles, I did find errors, which looked like this in the log file



**...**

**Pass 723466, frame 0, offset 00000000, time 908b5feb, errors 824483**

**ERROR: (E) offset 0000001E read FFFFFFFB desired FFFFFF7B.**

**ERROR: (E) offset 00000046 read FFFFFFFF desired 7FFFFFFF.**

**ERROR: (E) offset 00000084 read EFFFFFFF desired FFFFFFFF.**

**ERROR: (E) offset 0000008E read FFEFFFFF desired FFFFFFFF.**

**ERROR: (E) offset 000000B7 read FFFFFFDF desired FFFFFFFF.**

**ERROR: (E) offset 000000C4 read FFFBFFFF desired FFFFFFFF.**

**ERROR: (E) offset 000001B8 read FF7FFFFF desired 7F7FFFFF.**

**ERROR: (E) offset 000001BE read 7FFFFFFF desired FFFFFFFF.**

**ERROR: (E) offset 000001D2 read FFFFFF7F desired FFFFFFFF.**

**Pass 723467, frame 0, offset 00000000, time 90aea31f, errors 824492**

**ERROR: (E) offset 00000046 read 7FFFFFFF desired FFFFFFFF.**

**...**

# Conclusion

Since the errors in this flash are pretty well spread over random bits, and not in bursts (well, except for TODO), BCH codes are a better choice for error correction in these devices.

# References

<http://www.eeherald.com/section/design-guide/esmod16.html>

<http://www.eecs.berkeley.edu/~tking/theses/mshe.pdf>   
  
<http://www.chem1.com/acad/webtext/chembond/cb10.html>

<http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCIQFjAA&url=http%3A%2F%2Futrecht.ewi.utwente.nl%2Fsmi%2Fcontent%2Feducation%2Fis%2Fstand_alone_flash_memories.ppt&ei=udBzVP2lDtSeyASFzYHIAw&usg=AFQjCNHq7d8oJ9IVYwVk_XKo90s0zuZ1tw&bvm=bv.80185997,d.aWw>

<http://en.wikipedia.org/wiki/Electrical_resistivity_and_conductivity>

<http://hypnocube.com/products/controllers/hypnolsd/>

<http://hypnocube.com/2014/11/flash-endurance-testing/>

<https://github.com/ChrisAtHypnocube/FlashWrecker>

<http://siliconexposed.blogspot.com/2014/03/microchip-pic32mz-process-comparison-to.html>

[siliconporn] <http://siliconpr0n.org/archive/doku.php?id=azonenberg:microchip:pic32mx340f512h>

[PIC150] <http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC32MX150F128B>

 \* http://users.ece.cmu.edu/~omutlu/pub/flash-memory-voltage-characterization\_date13.pdf

 \* http://siliconexposed.blogspot.com/2014/03/microchip-pic32mz-process-comparison-to.html