PAUL HAGSTROM, KANSASFEST 2016

# DISKS, EDD, COPY PROTECTION, AND EMULATORS

# THE PLAN, SUCH AS IT IS

- Disks are aging, bits are rotting. A surprising amount of stuff is unpreserved.
    - Entire programs, but also versions, intro music, original title art. Many protected against copying.
- Hardware will also fail eventually. For preservation to have happened, it needs to be in emulation, stored in modern hard drives, backed up to the clown.

# UNPROTECTED 16-SECTOR DISKS

- Unprotected (5.25") disks are organized in 35 tracks of 16 sectors of 256 bytes each. DOS 3.3, Pascal, ProDOS, SOS. 140K (143360 bytes) of data, imaged into .DSK format.

- In a .DSK, the sector is found by looking in the right place. ((T x 16) + S) x 256 bytes in, you find 256 bytes of data. On a disk, you go to the track, read until you find an "I'm sector S" marker, then read encoded data and checksum, decode 256 bytes.

# CONFLICTING ORDERS

- "I'm sector S" and encoding is predictable, not part of the data but part of the container. DSK does not store that. Only stores 256 decoded bytes per sector, only location information for sectors is the position within the DSK file.

- Turns out what the disk thinks is sector S is not actually what DOS and ProDOS consider to be sector S. There's a mapping, and DOS and ProDOS differ. So DSKs can sometimes be ProDOS-ordered (PO) or DOS-ordered (DO). Have to know which if you want to find a sector.

# DOS 3.3 IS SO 1980

- Not hard to preserve unprotected 16-sector disks. COPYA will copy them. Every emulator can use DSK/PO/DO images. Hardware modern storage emulation (CFFA3000, Floppy Emu, SDFloppy II, ADTPro, ...) uses them (by inserting the predictable structural elements on their own).

- But change the container, and DSK won't work. Simple changes like making "I'm sector S" be "J'n tfdups S" and it all falls apart. Intentional protection, or even just 13-sector DOS 3.2.

# PRESERVE THE ACTUAL NIBBLES

- Imaging a simple structural change can be accomplished by reading the structural elements too. Don't fill in "I'm sector S" for the image, just record it into the disk image.

- Data is encoded on the disk, not in bytes, but in partial bytes (nibbles) that are combined upon reading.

- "Nibble copiers" make fewer assumptions about the container structure and just record what the Disk II interface card spits out when the disk is read.

- Most emulators can use this format too (.NIB).

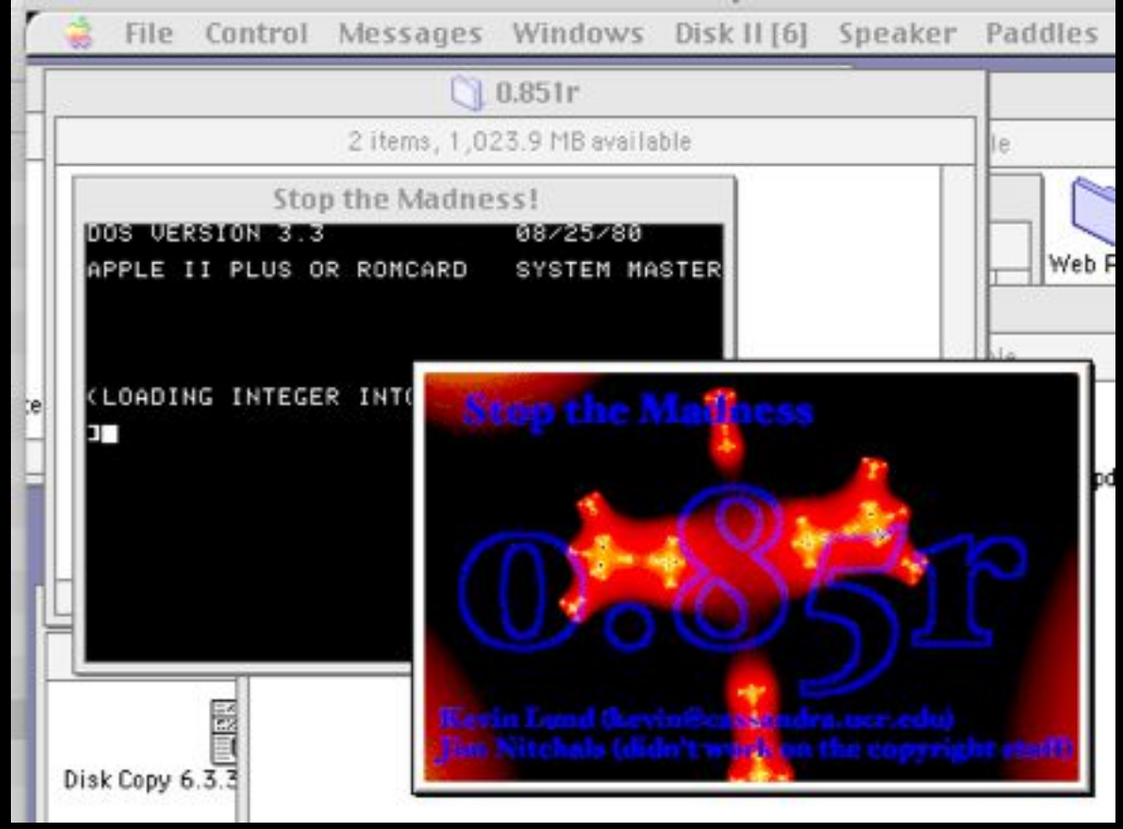
# WHY ARE WE NOT NOW DONE?

- Copying nibbles is still not enough to get around most intentional copy protection. For emulators, NIB files still make some assumptions. One is: there are 35 tracks, 4 phases apart, on which nibbles are stored. But there are at least 140 (35 x 4) different track positions the head can be moved to. If a disk has data on half- or quarter-tracks, .NIB does not store that.

- Can't just nibble copy all 140 quarter-tracks on real hardware, because writing on track 5.25 messes up data on tracks 5.00 and 5.50 at least.

- People protecting disks knew this very well.

# D5NI/V2D FORMAT




- One slight advance over the NIB format is the more flexible D5NI (or, in Virtual II, .v2d) format. Used (possibly first) in Stop the Madness.

- D5NI accommodates half tracks, is a nibble based format, and does not assume a constant track length.

- Definitely better than NIB for this, more likely to work more of the time. I've booted a still-protected D5NI image of Choplifter on Virtual II.

# THIS IS 2016. STOP MESSING AROUND.

- We are in a post-CSI world now. Stop messing around with nibbles and bytes, we can just record the magnetic flux on the disk.

- Kryoflux, DiscFerret, SuperCard Pro, all operate by sampling a (slightly more modern 5.25 PC) drive fast and recording the result. If there's magnetic information, it is detected and stored.

- There. Done. Here's your many MBs of flux data. You're welcome.

# MY KRYOFLUX IMAGED MY DISKS AND ALL I GOT WAS THIS LOUSY FLUX DATA

- Having a flux image is not the same thing as having a bootable disk. The hardware is years ahead of the software by now, the software has not been moving very quickly.

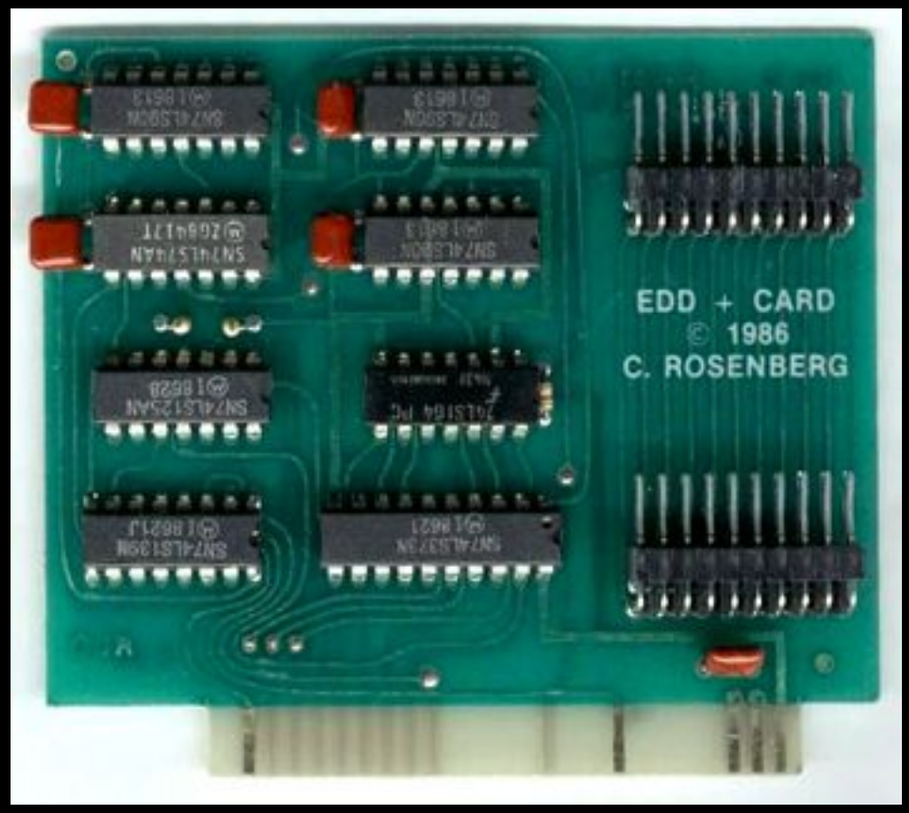
- You know what you can do with a flux image? Mostly, you can analyze it, and make a DSK image from it. If it's unprotected. And if the disk read was ok. And the disk read is not always ok, sometimes the head is dirty, etc.

- How very far we've come.

# MY KRYOFLUX IMAGED MY DISKS AND ALL I GOT WAS THIS LOUSY FLUX DATA

- (At the risk of spoiling the comic effect, MESS does have some capacity to handle flux images. I do not know if this works for the Apple II, or which flux image types it supports. Does anyone here know?)

# ESSENTIAL DATA DUPLICATION

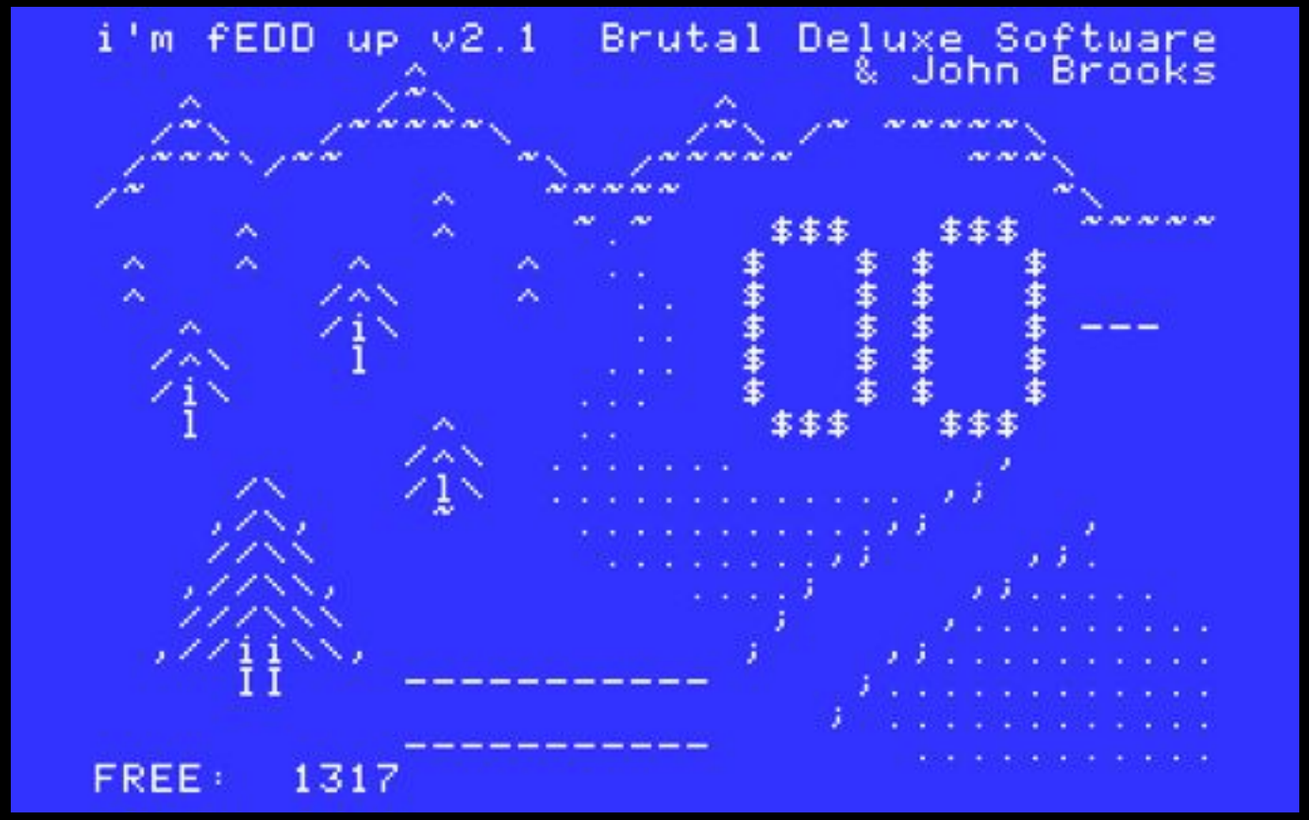



- In 1986, Utilico Microware made a peripheral card as a companion to the 4th version of their Essential Data Duplicator (EDD) software. This card is a little bit like a mid-1980s Kryoflux. Nowhere near the resolution, and it runs in a 1MHz machine, but it read bits faster and in more "raw" format than the Disk II card.

- In 2012 (I think), Brutal Deluxe released "I'm fEDD Up", which uses this card and records data onto ProDOS volumes in several formats including .EDD.

# WHAT IS IN AN .EDD FILE?

- The .EDD file contains a stream of bits as read from the disk.




- The Disk II has no way of knowing when the disk has spun around all the way (there is an "index hole" in the disk, but there is no detector wired into the drive), so I'm fEDD Up just reads the bits from a track for about 2.5 revolutions, then moves to the next track.

- Data looks like: 10110000000010100111101011110001

# WHY DO YOU NEED AN EDD+ CARD?

- Why can't you just read the bits from the drive without this extra peripheral card?
- The reason is that the Disk II card itself does some processing on the bits before it even hands anything over to the Apple II. The EDD+ card grabs the "preprocessed" bits, and grabs them faster, allowing for a more direct view of what's actually on the disk.

# FASTER THAN A SPEEDING BIT

- 1MHz is not many MHz. Like only half as many as 2MHz.
- Those bits are coming fast. The disk itself is rotating at 5 Hz and there are around 50000 bits flying by in that time. So about 250k bits per second. Or about 4 clock cycles on the 6502 per bit. Takes at least 6 to poll a data address.
- Disk II card runs at 2MHz, so has 8 clock cycles to deal with an incoming bit. The Logic State Sequencer collects the bits and presents them to the Apple II.

# LOGIC STATE SEQUENCER

- The Logic State Sequencer does approximately this:
    - Wait to see if we have a 1. Did we find one? Rotate it into the accumulating data byte. Did we wait too long without seeing a 1? Must be a zero then, rotate 0 into the accumulating data byte.
- There's a decision point: "It's been too long since the last 1, so this must be a 0." Pulses should come in every 8 clocks, but can be detected if it's between 4-12. At 12, LSS gives up and says no 1 is coming, this is a 0.

# ONES AND ZEROS

- The input to the LSS is just "something happened" or "nothing happened (yet)".
    - A something is a 1.
    - Nothing, when there could've been a something, is a 0.
- This makes it very timing dependent. It's pretty easy to lose your place in a string of zeros. (Given how long we've been waiting, how many 1s could we have gotten but didn't?)

# DESIGN DECISIONS

- Data written to the disk is written in bytes that always have the high bit set. This leaves only 7 bits for data, but has benefits. It serves as a signal that the data is ready, and allows a tight 6502 polling loop. (LDA C08C,X; BPL #$FB). It also allows for sync.

- LSS ignores any zeros it gets after the 8th data bit (including the initial 1) is set, waiting for the next 1 (since the next valid nibble is guaranteed to start with a 1). So FFFF... written as 1111111100111111111100... will quickly "self-synchronize" within five of those FFs.

# DOING MORE WITHOUT LSS

- The LSS ignores 0s between sync FFs (but they take time). So 11111111001111111100 and 11111111111111111 both look the same to 6502 code: one FF after another FF. The LSS discards the difference before any 6502 code gets to see it, but copy protection schemes can still depend on it.

- And a new byte shows up as quickly as every 32 cycles, insufficient time to decide whether a byte has taken too long to arrive. (LDA C08C,X; BPL #$FB takes 6-7.)

- The EDD+ card grabs the bits before the LSS gets them.

# WE'RE DONE THEN, RIGHT?

- So, the EDD file grabbed by I'm fEDD Up from the EDD+ card is a stream of bits, including any of these sneaky 0s hiding between valid nibbles, and in principle if we "play them back" (like, under emulation) the protections that rely on them should find them and work.

- Let's try it out.

# OPEN EMULATOR

- I made a bunch of EDD files, but at a certain point doubt crept in. I have these files. But did I successfully image the disk? How can I know?

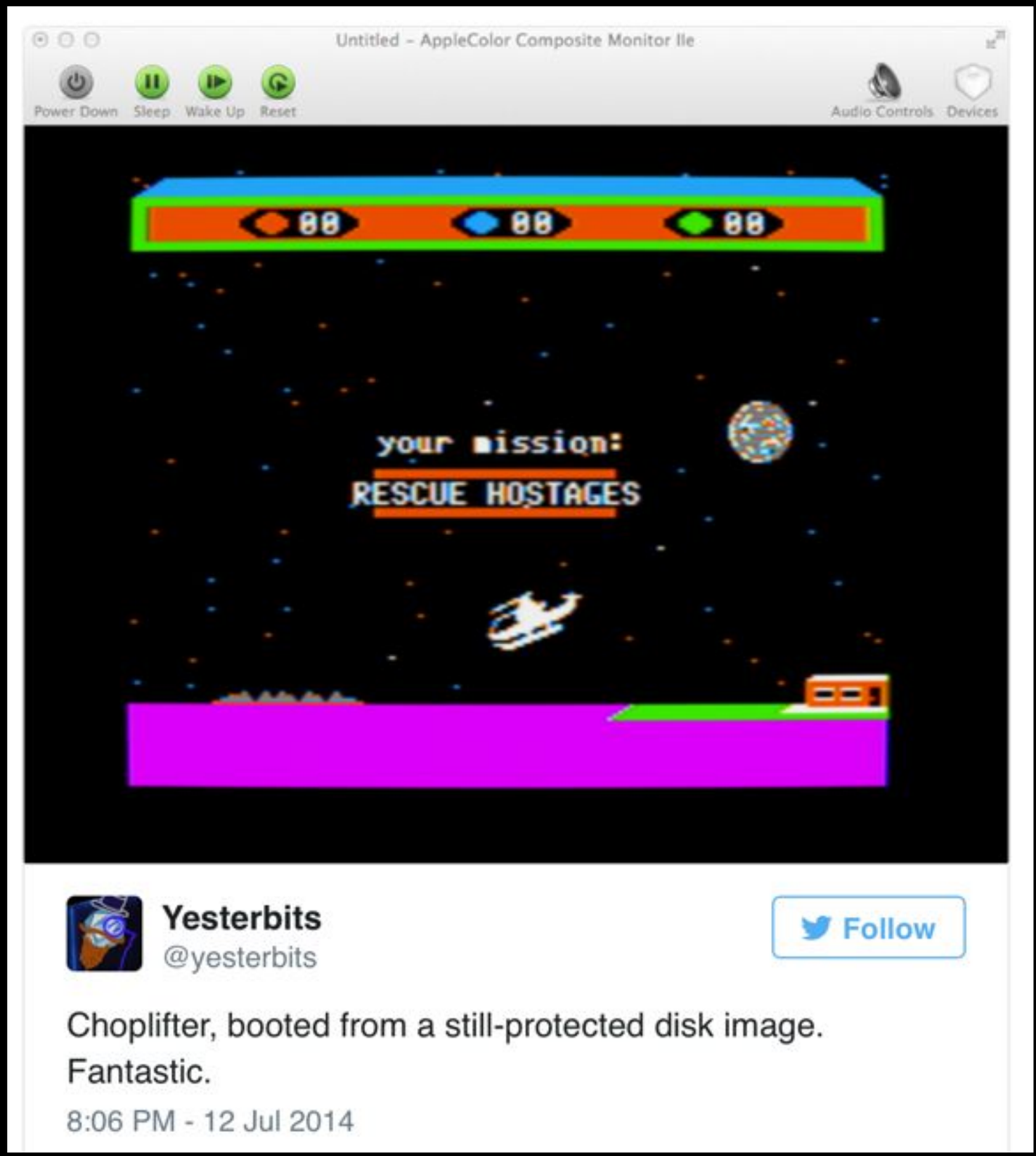
- I looked around for a while to find an emulator that would emulate the LSS itself, not just read nibbles from NIB images or bytes from DSK images and fake the results. Eventually I landed on Open Emulator, which reads a format called FDI (Floppy Disk Image) that has as one of its options a bit stream like the one the EDD+ card produced.

# DEFEDD

- So I wrote a little ruby script to take the I'm fEDD Up files and implant them within an FDI container. Not a difficult task, all that is needed is the addition of some header material. I called it defedd because it was undoing what I'm fEDD Up did, and I kind of liked that it was also a 24-bit hex number. Unfortunately, it doesn't have a very useful ASCII representation.

# FANTASTIC.

- And it worked! I now knew that my EDD image of Choplifter had actually captured the disk. Well enough that its own protection didn't know it was a copy.



# EXCEPT

- Lots of things didn't work. For a couple of different reasons. One of them is fairly easily described, that we might remember from the Olden Days: track sync.

- The Disk II has no sensor for the index hole, no way to know where it is rotationally on the disk. Some protections relied on finding their place on one track then moving to another track and reading things there. Relative angular positions were crucial for this. Some bit copy programs would try to keep tracks in sync in the same way, for this reason. Once EDD file is created, sync information is lost.

# THE ANXIOUS MC3470

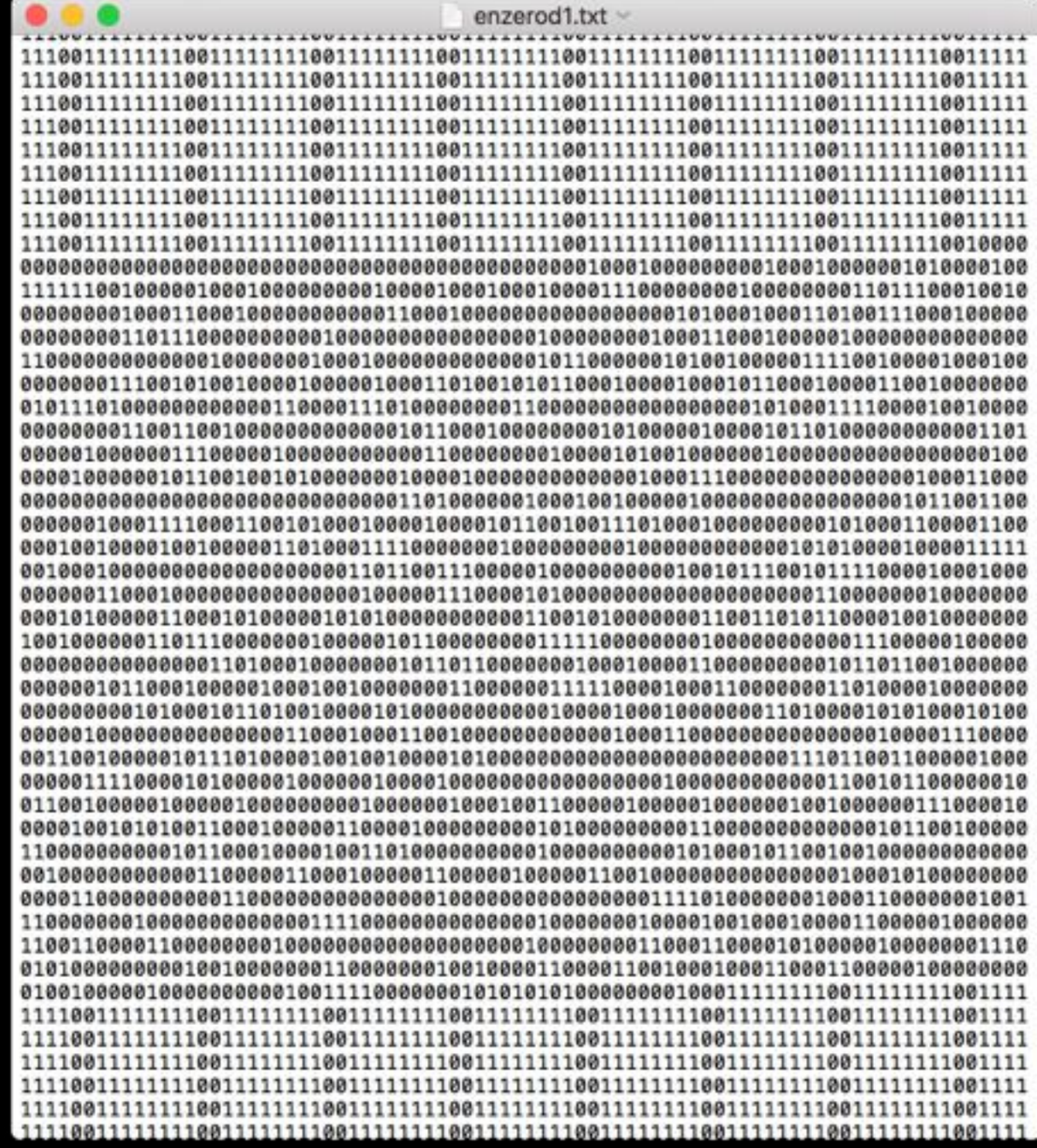

- The read amplifier (MC3470) on the Disk II analog board inside the drive is constantly gripped with anxiety over maybe missing something.

- A 1 is a something. Getting a 1 makes MC3470 feel happy. Not getting a 1 for a while can be called a 0. That's ok, MC3470 still feels happy.

- As more time elapses since getting a 1, MC3470 starts worrying that maybe there are supposed to be 1s but they're too quiet. So it starts cranking up the gain, desperate to find some 1s. And it will find them. Eventually. Even if by then it's just amplified background noise rather than an actual 1 signal.

# 000

- The MC3470 becomes unreliable quickly. It can read two 0s in a row, but after that, 1s can creep in.




- And this is I think the source of most of the trouble I have had working with EDD files. Those 2.5 track reads are not identical. Not identical in length (1s can be fast, only 0s are basically constant time), not identical in content. Finding the track is harder than it seems like it should be.

# NIBBLES

- DOS 3.3 solves the unreliability and zero timing problem by constraining the data it writes to never have more than two 0s in a row. The hardware itself can only read 7 bits per chunk (the 8th bit being a byte-start marker), and the in-principle-128 possible values are further constrained to those that have no more than two 0s in a row and do have two 1s in a row. This limits it to 64 possible values, so 6 bits of information per nibble.

- D5 11010101 and AA 10101010 are also used but only for the structural elements (not data). DOS 3.2 had only 32 available because it was limited to nibbles with not even two 0s in a row.

# BACK TO DEFEDD

- Since the original EDD-to-FDI converter, designed mainly to just verify that EDD images worked, I've continued to work on the defedd program. Off and on.

- An older version of this (now in Python 3, and quicker) is on github but it is a disaster, and soon I'll have a better version there. Was hoping it would be there by now, but looks like not quite. Wait for the new one though before trying to understand or play with it, I'd say. Only the EDD-to-FDI option works reliably in the one currently on github.

- But here's what I'm trying to do with it and where I'm at with it.

# DEFEDD: FORMAT CONVERSION

- First, more general format conversion. EDD to FDI works for Open Emulator, but this is the only emulator that I'm aware of that uses the FDI format. And OE only emulates up to the Apple II plus. Converting EDD to DSK and NIB (where EDD wasn't really necessary after all), and D5NI/V2D is one goal. Also can provide some overall disk diagnostics/analysis.

- Another is to target MFI (MESS Floppy Image) format so that these will work in MESS. Mainly because this is the only way to test images of protected Apple /// disks. MESS has limited EDD support already but it doesn't work very well because the problem is a hard one. If it were easy, defedd would be done already.

# DEFEDD: BIT REPAIR

- There are 2.5 reads of a track in an EDD file, so if one sample is bad, there are 1-2 other samples available. Also, tracks are "wide", so usually a track's data can be found (less reliably, but found) on the adjacent quarter-tracks. So there might be up to 12 samples of a single region of data, making it more likely possible to identify runs of "actual zeros" and recording them in the image file.

- This would be good for copy protection schemes that rely on the randomness, so long as the emulator also implements the MC3470 behavior. Open Emulator does.

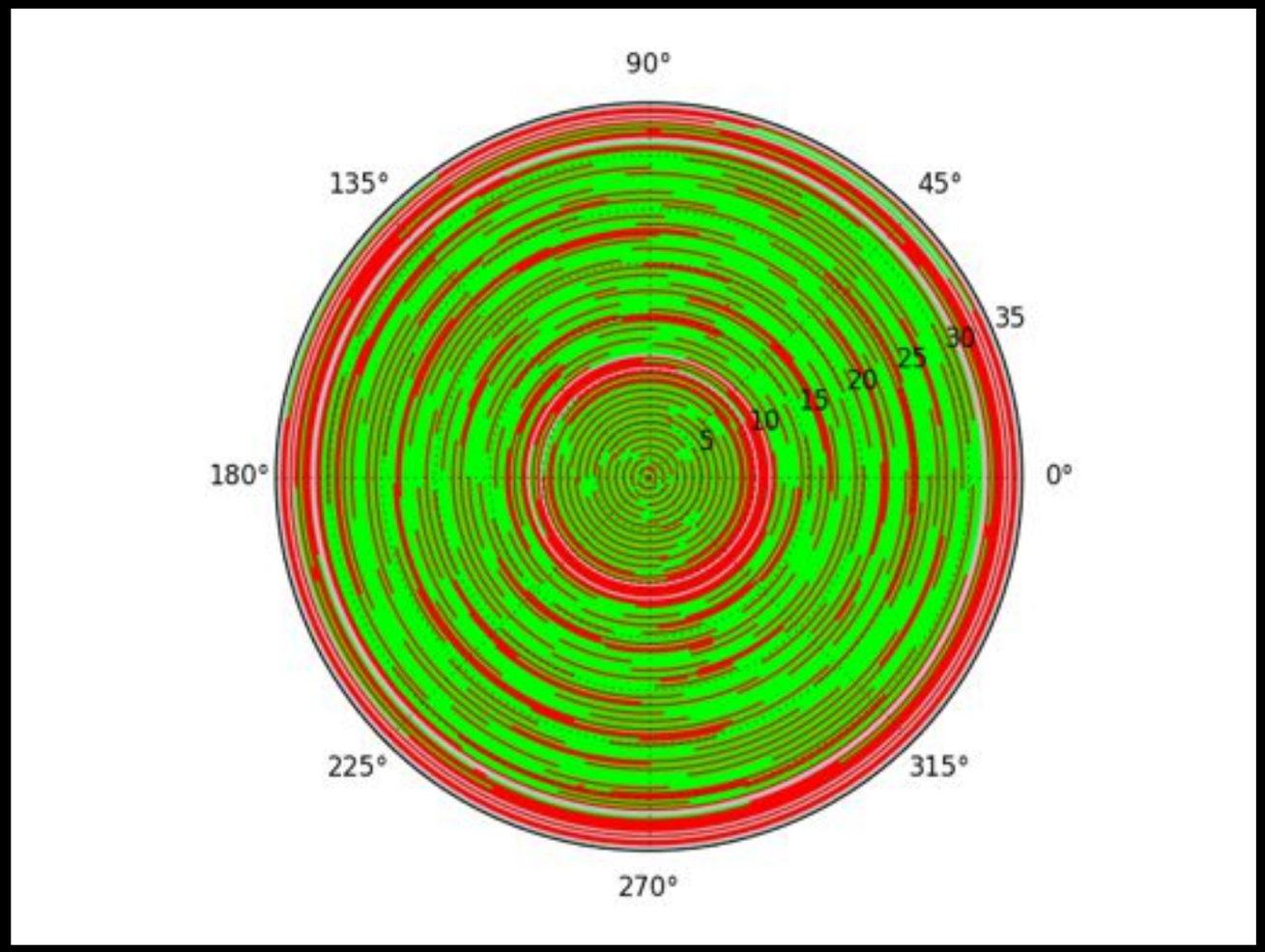
# DEFEDD: SYNC APPROXIMATION

- The "data bleed" between tracks should also allow for adjacent quarter tracks to by synced and possibly help make usable images for disks that rely on track sync.

- There is no usable crosstalk that I've found between tracks. Quarter-tracks can be synced together with their whole track, but track 1.0 cannot be synced with track 2.0. (Except, it turns out, if there's a spiral protection on the disk that links up the tracks more closely together.)

- For tracks where there is no real sync data, could approximate a spin based on observation of spin distance on tracks where there is sync data, but it's so highly variable that it's unlikely to work often.

# DEFEDD: ART




- I've also been playing around a bit with adding visualizations of the data to help understanding the results of the analyses.
- I think this will be useful in seeing the structure, but also: they look cool.
- (Initial inspiration/idea for this was Charles Mangin's program doing this for DSK images at last year's KFest.)

# ELSEWHERE

- There has been some discussion/work, I think mostly by John Brooks and Antoine Vignau, on enhancing I'm fEDD Up to:
    - read more at once, possibly even a whole disk—this would eliminate sync uncertainty, since the disk would continue to spin through the entire read across multiple tracks.
    - leverage the faster speed available on the Apple IIGS as well as a direct-read mode in the IWM chip to detect timing bits without the assistance of an EDD+ card.

# IMAGE VERIFICATION

- Worth noting that not all my EDD rips worked. Some had to be done again. At one point, my MC3470 chip itself just failed and I didn't notice immediately, but my EDD files were filled with just 0s. There needs to be a way to check to see if a read succeeded.
- Booting the disk in Open Emulator is a huge step in that direction (and that was the original point), but all it can tell you is if it did NOT work. If the disk boots, it's still not certain whether some other part of the disk image is corrupt. Analyzing the image can at least help with confidence that the rip does not need to be re-done.

# PRESERVING THINGS

- The holy grail in this area of preservation is to be able to image a protected disk and run it, protected, in an emulator. If it works and can be fast enough, we have a better chance of archiving all those programs (and versions) that are still out there unpreserved in the twilight of their magnetic years. Allows archiving without cracking, first of all, and also allows people of the future to try their hand at cracking as well. And it preserves the copy protection scheme itself as software.

# THINGS I FORGOT TO INCLUDE