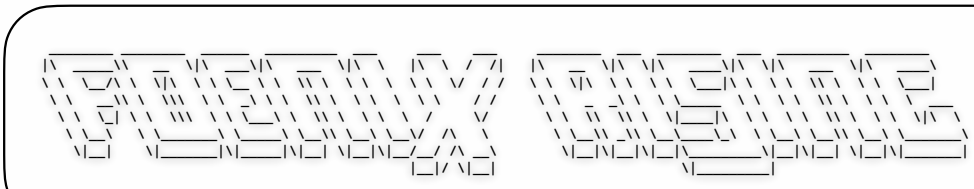
(44 years, 2 months, and 10 days after the inaugural "The Transactor" bulletin release, Foenix Rising is here)



commentary and information
concerning your
FOENIX RETRO SYSTEMS

ISSUE #1
July 10, 2022

**This is the first news bulletin which will be issued up to 12 times a year for enthusiasts of retro technology and specifically, Foenix Retro Systems users throughout the world.**

**You may have been auto-subscribed to this bulletin, but may opt out at any time (see below).**

**Whether you own a Foenix now, or intend on purchasing a Foenix in the future, you may find this a valuable resource to keep up to date, including:**

1. **Information on software available**
2. **Information and specifications on hardware, add-ons, & modifications from Foenix Retro and elsewhere.**
3. **Information on problem shooting (also known as "troubleshooting" :) and Foenix FPGA and kernel releases, and a variety of other information.**

**Below is a free copy of our first issue.**

**Now we have some good news.**

**The Foenix Marketplace (AppStore) will soon be available and will provide downloadable apps, utilities, code samples, and binary data released publicly, or from the Foenix Rising publication.**

**To unsubscribe from Foenix Rising, please send an email to: commerce@emwhite.org with a subject of "unsubscribe".**

**Thank you for your support and interest,**

**Michael (EMwhite)**

---

I would like to thank Stefany Allaire and the Foenix community for being welcoming, and for entrusting me to publish this newsletter. I would also like to thank those that contributed to, and offered feedback on issue #1.

Most of all, I need to acknowledge the publishers and contributors of The Transactor, the original TPUG News, TPUG Magazine, and published content from user groups that began in the late 70's and early 80's of the last century.

Well before the world-wide-web, information managed to make its way to the far reaches of the world thanks to efforts of people involved in these endeavors. We stand on the shoulders of giants. The retro-style layout of this insert is in their honor.

This page is mostly blank

Foenix Rising

A newsletter for developers, users, and enthusiasts of Foenix Retro Systems products

Issue #1 July 2022

## Hello Foenix

Welcome to the inaugural issue of *Foenix Rising,* a Newsletter for Foenix Retro enthusiasts.

Why a Newsletter and not a 'blog'?

30 years young, the web is still difficult to navigate; despite the efforts of archive.org, nothing on the internet is permanent, and though we all try, we've not come close to escaping the wrath of ads.

Thankfully, we can still get our hands on .pdfs or bitmap scans of mags such as Transactor and other classic 80's and 90's Computing Journals. For as long as filters are able to render ancient formats, content like this will be viewable and printable.

If you must 'print', please do so responsibly, and store it for reference and discovery by future generations of our kind.

What to expect: a mix of Foenix updates, developer published technical articles, activities, interviews, and a few surprises; the aim is to inform, entertain, and enrich your Foenix experience.

Foenix Rising is currently a one person endeavor, but the hope is to publish content and code examples from members of the community. Thank you for your support.

-EMwhite

## Featured Photo - The A2560K

The 'business end' of the A2560K, Serial #003. This pre-production unit was hand-assembled with love in British Columbia, Canada.

To be accurate, the 'K' is all 'business'; every angle, profile, texture, feature, and the design decisions behind them were optimally engineered to attract and inspire expert engineers and new users.

---

**VTOC** (volume table of contents)

## `git` and URL Resource Directory

Updated each issue, this space will contain links to public facing Foenix related development efforts (useful to see how others have solved particular problems, to find docs or to see what is out there)

| | |
|---|---|
| Env | https://github.com/WartyMN/A2560-FoenixRetroOS |
| Lang | https://github.com/daschewie/FoenixBasic68k |
| Kernel | https://github.com/pweingar/FoenixMCP |
| Compiler | https://github.com/hth313/Calypsi-m68k-Foenix |
| Env | https://github.com/Trinity-11/FoenixIDE |
| Game | https://github.com/dtremblay/fraggy |
| Code | https://github.com/clandrew/experiments |
| Game | https://github.com/hth313/defender-Foenix |
| Compiler | https://github.com/hth313/Calypsi-65816-Foenix |
| Samp code | https://github.com/noyen1973/C256-Foenix |
| Utility | https://github.com/econtrerasd/Foenix-Sprite-Editor |
| Game | https://github.com/dtremblay/c256-tetris |
| Utility | https://github.com/dtremblay/c256-vgm-player |

Links to other Foenix Resources:

Foenix Retro Systems
Home Page

Foenix
Discord Invite

Stefany Allaire
Patreon Page

VCF East 2022 Foenix
Exhibit Retro

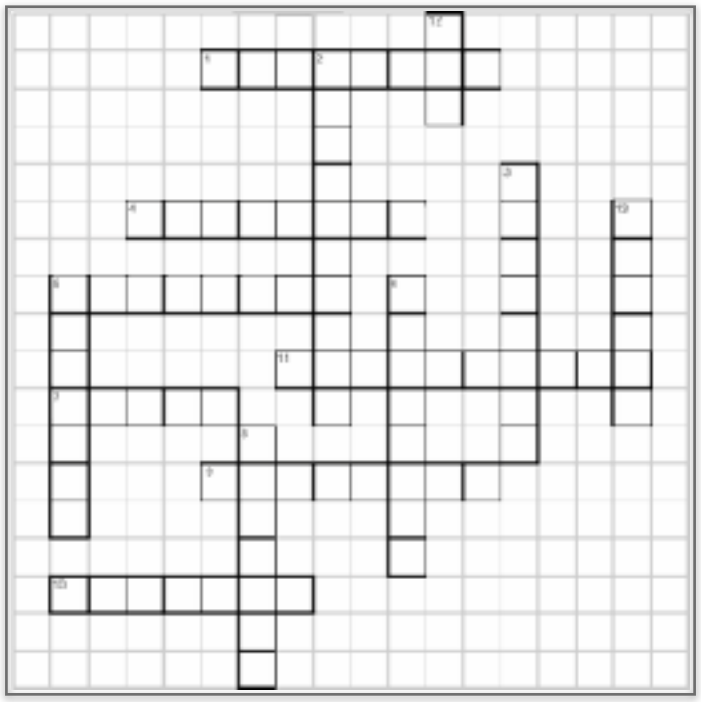
*Foenix Rising* is a user-supported, not-for-profit bimonthly hobbyist's newsletter published in Murray Hill, New Jersey, USA supporting Foenix Retro Systems products with a focus on software development and related retro technologies.

Subscribers: 966

Published by EMwhite (discord and elsewhere)
Motto: 'Beware of programmers with screwdrivers'

Correspondance: commerce@emwhite.org

# CROSSWORD



## Theme: Tech Pioneers and Visionaries

Test your knowledge (and researching skills) of historic innovators by last name - Googling™ permitted

| Across | Down |
|---|---|
| 1 Mario's Nintendo father, credited for saving gaming | 2 `www` HoF member and NCSA wunderkind |
| 4 Chuck E. Cheese founder and pong daddy | 3 Macintosh GUI King and Hypercard inventor |
| 5 First name shared with Peanuts Lucy's brother | 5 "Jack Attack", known to market to the masses |
| 7 _ _ _ _ _ 2049'er and Lorraine founder | 6 Knighted Clive partnered with a US watch company to introduce a doorstop |
| 9 Lady Ada's (not Limor Fried) surname | 8 D-list' partner and Sharks 'super-fan' |
| 10 Lang gifter and National Medal of Technology recipient* from Bell Labs | 12 Responsible for making 'Ed' visible. Co-founded a Unix workstation giant |
| 11 Cubist inventor of W3 (suppress the hyphen, if there is one : ) | 13 Root domain admin, SMTP RFC editor, and 'g-d of the internet' |

*presented to 'this' person and an esteemed colleague by U.S. President William Jefferson Clinton in 1999

# Interview with Peter Weingartner

Kernel Architect and Developer of MCP, the FMX / C256 and A2560 Kernels, and BASIC816

Fresh from committing version 1.0 of the A2560K MCP Kernel, Peter was kind enough to sit down and play 20 questions.

This turned into a 2 hour discussion across which we talked about Foenix platforms & Software Engineering; but also, Education, Astronomy, Peter's proudest accomplishment to date, and what's next for him.

For those unaware, MCP* (or Master Control Program) is a combination BIOS, Kernel, and command line interface, written from the ground up, primarily in 'C' for the A2560K. MCP will eventually be ported to the A2560U platform, the A2560X platform, and beyond. More on this later.

---

EMW: Thank you for making yourself available. Now that 1.0 is in the books, how are you spending your spare time?

PJW: I've starting to work on some of my tutorial videos again. I've just completed the audio for one on interrupts.

EMW: I'm actually a customer of your videos and am looking forward to that. Let's talk about your background. What was the first computer you used or had access to, and what was the first computer you owned?

PJW: First used was an Ohio Scientific 6502 based computer with no graphics outside of the character set. It was black & white. This was in Middle School.

The first computer I owned, and I still have it, is a Commodore VIC 20. It took years to save up enough Christmas money to buy it.

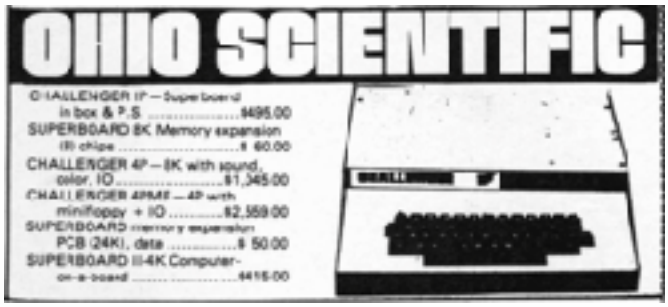
EMW: How long did it take you to buy the 1540 or 1541 floppy drive?

PJW: I guess 40 years because I still don't have one. I was too poor and relied on the Commodore Datasette primarily. Well wait, that isn't true, a neighbor gave me a Commodore SX-64 which I still have so I guess technically, that was my first disk drive.

EMW: Our backgrounds sound similar, the first computer I used was a Commodore PET, also in middle school. Did this early start lead to Computer Science in College?

PJW: I started as a Physics major but around Junior year of College, I looked to take an Operating Systems class, just out of interest. There was a math pre-req and I approached an advisor to try an obtain a waiver where I was told that with the addition of just another class or two, I could graduate with a double major; Physics and Computer Science. So that's what I did, entirely by *accident*.

EMW: That's a first (!!), typically it's the opposite; I've heard of College Advisors unintentionally misleading students to the point where they miss graduation after being unaware of a late requirement. In your case, it sounds like it worked out.




EMW: What led you to Foenix and what were you doing prior?

PJW: For years I've wanted to make my own single board computer [*walks to closet, returns proudly with a wire-wrapped protoboard*]. 4-5 years ago I began. I got the terminal working, got EH Basic working, and started looking around for graphics chips and found nothing.

Sure, there were old VIC chips out there, but I wanted something that could do VGA or DVI-I and didn't want to do the heavy lifting myself from scratch.

EMW: [interjects] This was prior to Ben Eater's "Worst Video Card" series?

PJW: Yes. But I was watching the 8-Bit Guy [David Murray] and he was talking about his dream computer project and I thought that perhaps I should start working in that direction.

David was talking about Stefany and her project in the early days. At that point I started following Stefany and she posted something mentioning that she had a hardware design in the works but needed some help writing the software. So I approached her.

---

* Though we did not talk about it, MCP is a *Tron* (movie) reference. See footnote '1' on pg. 2 of the MCP documentation for another.

PJW: The earliest version of the Kernel was originally designed by somebody else along the lines of the Commodore 64 Kernel (even to the extent that it had common entry points such as SETNAM), but he didn't get very far with it.

I began playing around with the IDE and then started working on the Kernel and a BASIC interpreter.

EMW: So this was for the original FMX, I assume. How long have you been working on MCP for the 68K processors or for the 'K' specifically?

PJW: Just about a year; the first commit was on August 27th, 2021 but I started the code between June and July.

EMW: How much of the MCP is Assembly language?

PJW: Not much. There is a platform specific startup file that gets called upon reset and there are stubs for the interrupt handlers that call into the 'C' code and a few other little things; it's really localized.

EMW: Which complier was used for MCP?

PJW: I used the VBCC complier. I tried GCC but couldn't get the linker working the way I wanted. I'll probably move to Calypsi with the next major release.

**Peter shared the source statistics which represent the committed v1.0 code:**

**C code: 44,849 lines**
**Assembly: 418 (including a single startup file)**
**The assembled Object file = 241,196 bytes**

**Calculated, this is just less than 1% asm**

EMW: I've never seen or used the A2560U, but there was certainly a lot of excitement about Stefany breaking into the Motorola processor family. What Kernel did the A2560U use?

PJW: It was an early version of Foenix MCP. My hope was I could keep the code in sync between the two platforms but not having a physical A2560U, I quickly realized that without hardware, it would be impossible to continue development on the main MCP platform for the 'K' while confirming full compatibility with the 'U', so Vince has taken up working on the back-port of MCP to the A2560U.

EMW: Considering your experience with the FMX line (and by proxy, the C256U) and your work on MCP, what is your favorite Foenix machine either released, in production, or on the drawing board?

PJW: At the moment, it's the 'K'. It's about the case and the physicality of it more than the internals or electronics.

It's nice having a 68040 in there but one of the things that I'm realizing now that I've released MCP is that the 68K is kind of boring because the instruction set is so orthogonal and clean. Whereas the 65816 is kind of quirky and you have to really think about assembly language; it's frustrating when it goes wrong.

EMW: [interjects] So the 65816 is enjoyable when you've overcome some of the complexity or limitations?

PJW: Correct, Stefany has done a really good job with the 'K'. She teased me for such a long time "one of these days, I'm going to make a 68000 based, all-in-one keyboard computer" and I was like yeah yeah yeah, one of these days, you'll get to that!

EMW: And here we are, she's done it. Come to think about it, it's been just 3-4 months between her proudly holding up the prototype during the announcement at VCF East in October, and shipping the hand-assembled unit you show in your video, then another 4-5 months to where we are today, pick-n-place built units, just about ready to ship.

If you had 3 more months and more importantly, 3 more months worth of energy, what would you add or change about MCP?

PJW: I might not do anything more with it. Stefany wanted a code base that is a very simple stock kernel. We didn't want any memory managers or screen managers, or multi-taking, so that's not on the to-do list.

EMW: Yes, I recall reading in your MCP doc, goals and anti-goals state this clearly.

PJW: I had some thoughts of fancier ways of organizing user code, similar to the resource fork of the Macintosh. At present, we are not supporting partitions on the hard drive, so that could be added.

I feel like it's pretty close to what it should be. If I were writing an OS for myself, it would be a different matter. But we were trying to present a particular environment.

EMW: Taking everything into account, your academic background, 17 years at Intel, and your work with Foenix or something else that we haven't discussed, what would you say is your proudest accomplishment?

PJW: The first thing that comes to mind is something that I did 30 years ago. When I was a graduate student I got really interested in the PostScript language. Upon learning it, I realized there were not many resources, so I sat down and wrote a guide to the language with examples and an explanation on how the different operators worked and I posted it out on the web; this was around 1993 or

1994. I think we were still dealing with Mosaic 1.0 at that time; these were early days.

I just put it out there and it really took off. People really read it, I received requests for permission to translate it and somebody was interested in putting it onto a CD-ROM of PostScript utilities. Even as recently as 4-5 years ago, I would receive email with questions and 'thanks'.

**For those interested in Peter's PostScript doc, have a look here (fifth edition, February 2006 update).**

**Editorial note: I purchased a PostScript book in the early 90's and have messed around with Ghostscript over the years. By any measure, Peter's doc is an impressive piece of work. Worth a browse whether or not you know what PostScript is.**

EMW: If you could write a programming language for Foenix or port an existing language, what would it be?

PJW: Two things that I was thinking about; one is FORTH; it's both very fast and very powerful but I'm not sure how many people would use it. The other thing I'm thinking about is LOGO which is kind of a LISP dialect as it lets you define words and has good list processing; the Turtle graphics which would be kind of fun as I would leverage VICKY II and support Sprites, etc.

EMW: Would you choose to do a clean-room version of LOGO or would you port existing code?

PJW: I thought about one of the University of California Open Source versions but looked into it and it appears to be more POSIX based so I might consider a clean-room implementation.

EMW: I'd use that. I learned LOGO and PILOT as a kid, it's addictive and may even draw more people to the platform.

Ok… time for the lightning round. What is your favorite food?

PJW: Thai food. Specifically, Green Curry Tofu.

EMW: What was your favorite job, either in IT or something unrelated. I'll tell you mine: I drove a Produce Truck one summer and loved it. Also, I got into about seven accidents that year between the Produce Company and a stint at Domino's Pizza, delivering. This was before they discontinued the "30 minute guarantee"!

PJW: In grad school, I was a TA (Teacher's Assistant). I also worked in the Computer Lab at that time. I really enjoyed working with the students and helping them solve problems.

EMW: Do you know any written or spoken language other than English?

PJW: Well, I can say "Where is the tea" in several languages but that's about it. I did study Latin for 5 or 6 years, however.

EMW: What is your least favorite Programming Language; perhaps one that you wish you could *unlearn*?

PJW: [laughs] BASIC! It's such a primitive language. I used it as a kid. I suppose it's good for small things. When I was n High School while getting ready for the AP exams, I read some of *Metamagical Themas* by Douglas Hofstadter and was really taken by how elegant LISP was. I subsequently tried to write a LISP interpreter using Apple Pascal. That didn't work out so well!

EMW: Is there a person you most admire?

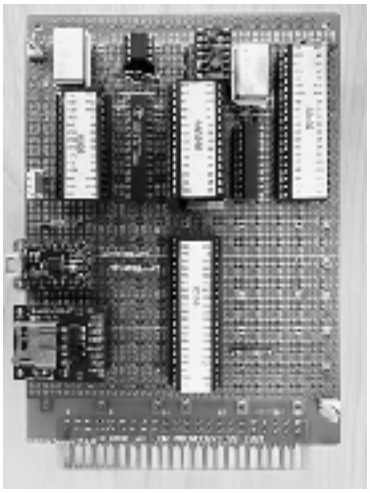
PJW: This isn't a person that I admire the most in totality, but in terms of technology, I'll say Steve Wozniak. He is one of the most interesting people and *his* Apple was the one I most liked. His version was the "pull the case open, here are slots for expansion, and here are the schematics", whereas Jobs version of Apple was "we need to put screw heads on here that nobody has a screwdriver for".

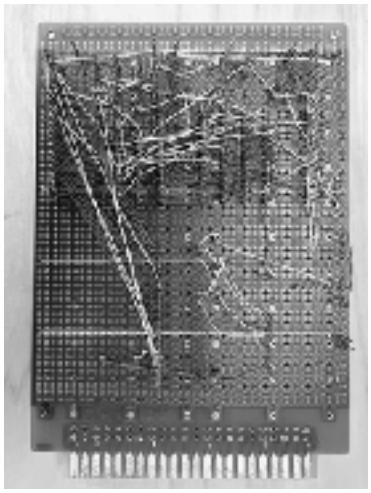
EMW: I couldn't characterize it any better than that! Thank you for the chat, this has been fun.

---

Peter's last response said it all. From his earliest work in College through his most recent `git commit` and onto his "Tail Recursive" YouTube videos, Peter is all about open platforms and 'community' and we thank him for all that he has done and continues to do for Foenix and its users.

**Here are a few pics of the wire-wrapped SBC that Peter spent time on prior to diving into everything Foenix:**



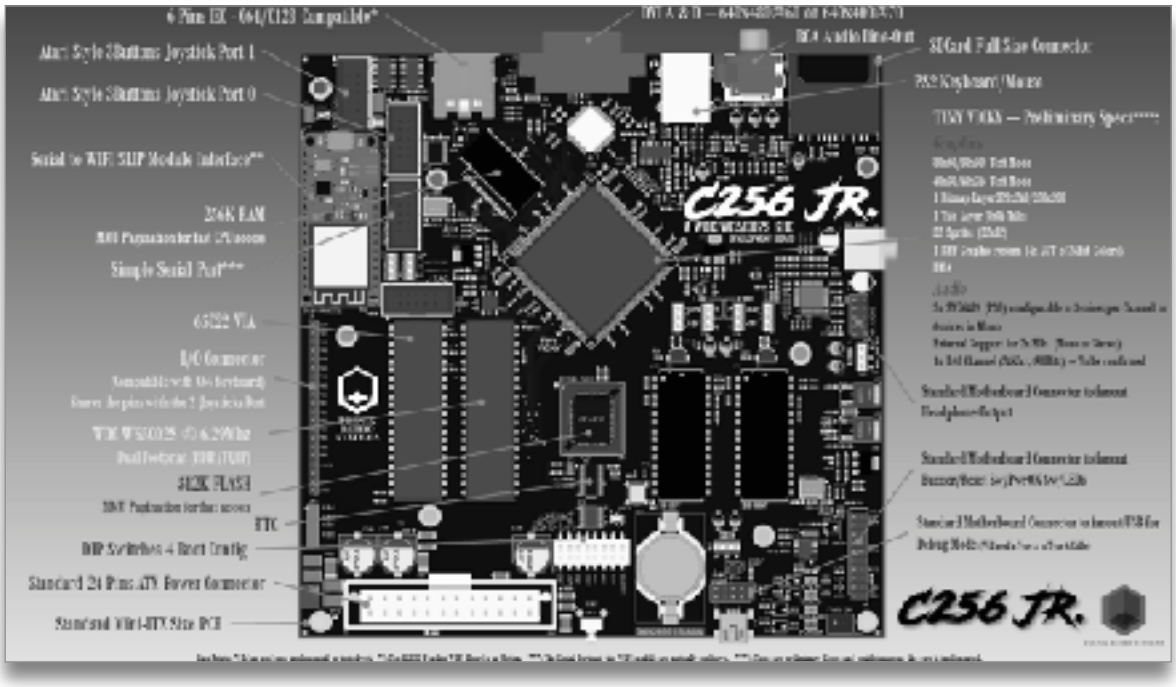


# C256 JR. - On the Drawing Board

Proof-of-concept details of an exciting new development board released

With several mid to high-end platforms in varying stages of build and early customer ship, Foenix Retro did something unexpected last month; it released details of yet another new system. This time, it's a full featured, but low-cost platform that looks to capture a segment of the retro market that we've not seen addressed.

The physical specs of the C256 Jr. board include an IEC (Commodore Serial) port, familiar Foenix staples (a VICKY derivation coined "TINY", sockets for stereo SID ICs, DVI-I video output, a real-time clock and other features) and a genuine WDC 65C02S with what looks to be a banked memory scheme many times the size of what you would expect from an 8-bit system.




So it looks like some of what we've seen before but in a lower cost and interesting package. But look closer, and you'll see that the platform has some new-to-Foenix features as well, namely, a Serial to WIFI SLIP interface intended to get the platform on a local private net with a minimum of hassle.

The left side of the board contains another nod to the early MOS lineage, a DIP packaged 65C22 VIA which brings the familiar Commodore 20 pin keyboard header and joystick interface to add to a set of PS/2 keyboard and mouse ports that Foenix systems have been using since the original FMX.

The image released above represents the development board of which a small quantity are being produced. These are being put into the hands of developers and supporters that will test the new hardware and work in parallel to produce kernel and operating environments. Two kernels are expected to be available for the Jr. The first is a clean-room re-implementation of the CBM "KERNAL", suitable for running a user-provided CBM BASIC ROM image with a Commodore disk drive. The second is a multi-tasking kernel (ported from the 65816 machines) with networking, SD Card support, and a high performance byte-code compiled BASIC.

While preliminary, Stefany Allaire has hinted that the prod release of this SBC is expected to be less than $199 USD (BYO case and power). Final specifications and the bill-of-materials will be refined across the next few months.

It is expected that a fair number of C256 games and applications will be ported, and the possibility exists that the prod release of this board will host 'other' platforms.

Something else unexpected: for the first time, a Foenix platform built to the Mini-ITX form factor with support for 24-pin ATX power; this will be familiar to PC and kit builders and opens the door to a world of options and customization, not the least of which could be a paring with cases such as the *My Retro Computer* Mini-ITX case (see Kickstarter campaign from earlier this month).

Since we are still in the 2020s, supply-chain uncertainty for some components may complicate matters; thankfully, with a fair number of discreet ICs on the board, FPGA logic cell requirements are more modest than demanding, and our understanding is that the dev board will run comfortably with well less than 16,000 cells. More will be known as the final specs and BOMs come together.

So move over diaper-boy DK and Charlie Chaplin 'on-a-budget', there is soon to be a new 'Jr.' on the scene.

# The Machine that Builds Machines

Foenix Retro adds powerful pick & place capabilities with the Charmhigh Tech CHMT48VA

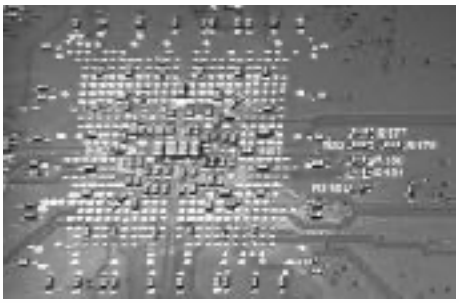
If you've been following along at home, you'll know that Foenix Retro recently added a new tool to the arsenal; a Charmhigh Tech pick & place (P&P) machine. This article takes a quick look at the technology and what it means for Foenix Retro.

The CHMT48VA (or '48VA', because we are into the whole brevity thing), places components onto PCBs at a rapid rate and with high precision, accurate to within 25/1,000ths of a mm. Parts are fetched with the aid of a vacuum nozzle from a bank of 27 reels mounted on the left side of the machine.

An embedded Linux system drives the robotics according to instructions output from the same Altium Designer software that produces the PCB layout and the solder mask stencil. Despite what we know about CNC, robotics, and the capabilities of 3D printing, this application of related tech still seems futuristic. Seeing it in action is impressive.

Surface Mount Technology (SMT) came into prominence in the mid-to-late 80's and is simultaneously thanked and cursed for efficiency, innovation, cost reduction, eye strain, and back pain.

Those of us with through-hole soldering skills usually cry uncle when asked to hand solder even a handful of SMT passives to a PCB.



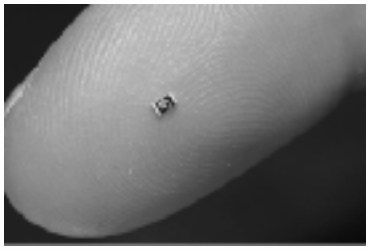
Go ahead and try placing 1,000 components on a PCB and I'll introduce you to my optometrist; how about 10,000 components across a 10 hour production day?

To illustrate just how challenging this is, try to simply count the capacitors and resistors in the picture above. This is a pic of the back-side of a C256U+ board beneath the Cyclone IV FPGA footprint. All of these caps and resistors were placed by-hand with tweezers, by Stefany. Had a P&P been available, this task would have take under 2 minutes.

In 1988, Steve Job's NeXT Computer company produced a video entitled "The Machine to Build the Machines", a Hollywood style production that introduced P&P and SMT to the general public. The 6 1/2 minute video was shown at the NeXT unveiling event in San Francisco, California.

The machine in the NeXT video is quoted as being able to place 150 components per minute which is about 3x the speed of the Charmhigh; but of course, the 48VA is likely 100x less expensive, and therefore, accessible to serious makers and small scale production outfits.

Stefany's 48VA is busy building Foenix A2560K machines and is able to place about 80% of the mainboard parts in about 20 minutes, directly from 8mm reels. This includes "0402" (0.040" x 0.020") and "0603" (0.060" x 0.030") components, used heavily on Foenix platforms.



For scale: Stock photo of a '0603' 300nF capacitor

Components of this type include resistors, capacitors, diodes, and SOT23 transistors. The 48VA comes equipped with two nozzles and can be adapted to place larger tray based components.

The following video, hosted on Amazon from a Charmhigh seller, shows the slightly bigger brother (the '48VB') placing parts, some of which are being fed by a human from rectangular trays towards the front of the machine. Keep a sharp eye on the front leftmost IC around 00:37 seconds in. (be careful not to blink)

Notice that upon 'picking', the arm swings over to an upward facing CCD camera. This crucial step confirms orientation and measures skew to within within a fraction of a millimeter and will compensate to more accurately place components to the pre-programmed locations.

One unavoidable physical challenge shared with 3D printers is oscillation caused by stepper motors, servos and acceleration / deceleration of the head. P&P machines add the additional challenge of noise and vibration caused by the vacuum pump. To counter this, Foenix Retro's P&P machine happens to sit on a concrete block foundation.

With a busy summer ahead, the 48VA will join forces with the Voronator 3D printer, solder reflow ovens, and all of the manual toil required to design, build, and test the rest of the works; this includes hand soldering and assembling connectors, headers, audio components, cases, diskette & hard drives, SD card sockets, and hundreds of other components. As those watching closely have come to appreciate, it's not just having access to machines and parts that produce an excellent product, it's knowing how to best use them. More goes into Foenix Retro products than meets the eye.

# Beginner's Corner: "Up, up, and away"

Pg. 71 of the *C64 User's Guide*, revisited

**Sprites, then and now**

Introduction to this column: In this first issue of *Beginner's Corner*, we will be leveraging BASIC816, simply because it's accessible to ~70% of the Foenix platforms that are in user's hands today. Next month, we will transition briefly to 65816 assembly, in order to tie some of the concepts covered (and a few others) to the good work that PJW has invested in his 65816 videos.

Longer term, we will land on something platform independent; may be the C Language or possibly a new multi-platform version of BASIC. With multiple efforts in the works, we expect to know more by the end of the Northern Hemisphere summer.

Personally, I'm a big believer in high-level (especially interpreted) languages for educational purposes because they promote experimentation and learning without requiring complicated tool chain knowledge, mastery of a language, or fear of losing work when things go 'black'. BASIC was, is, and until further notice, will continue to be, a reasonable tool for beginners.

If *retro* = nostalgia, you'll be hard pressed to find something as nostalgic as the famous 'Commodore Balloon Sprite' demo, which was (for many) a first taste of the power of Sprite Graphics, and one of many features that put the C64 on the map.

Yes, Atari had Player/Missile graphics and of course, Apple (part of the 77' Trinity) was first to feature a color [but non-linear / difficult to manipulate] bit mapped display; but Commodore took a fair amount of heavy lifting off the shoulders of would-be developers (and the 6510 processor) by creating a custom ASIC to deal with the headaches of managing and manipulating layers of movable objects, controllable via a relatively easy to understand register scheme.

Page 71 of the C64 User's Guide included a short BASIC program that out of the box, provided near instant gratification with just 5 minutes of typing. The curious among us modified the code to change the color, speed, size, direction, and in some cases, the actual image from the data statement defined hot air balloon you see on this page.

Upon typing `RUN` and pressing <RETURN>, something wonderful happened, and I'm not referring to the balloon. It was a wondrous new sensation followed by a realization: "I did that !! I can program a computer !!".

```
1     REM UP, UP, AND AWAY
5     PRINT "{CLR/HOME}"
10    V = 53248 : REM START OF DISPLAY CHIP
11    V + 21,4 : REM ENABLE SPRITE 2
12    POKE 2042,13 : REM SPRITE 2 DATA FROM BLOCK 13
20    FOR N = 0 TO 62 : READ Q : POKE 832 + N, Q : NEXT
30    FOR X = 0 TO 200
40    POKE V + 4, X : REM UPDATE X COORDINATES
50    POKE V + 5, X : REM UPDATE Y COORDINATES
60    NEXT X
70    GOTO 30
200   DATA 0,127,0,1,255,192,3,255,224,3,231,224
210   DATA 7,217,240,7,223,240,7,217,240,3,231,224
220   DATA 3,255,224,3,255,224,2,255,160,1,127,64
230   DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
240   DATA 0,62,0,0,62,0,0,62,0,0,28,0
```


Gotta love* the vintage color palette !!
(320 x 200 resolution - for ref/scale, each character is 8 x 8 pixels)

The Commodore 64 example was made possible thanks to the C64's VIC-II chip which supported 24 by 21 pixel single-color Sprites. Each Sprite required 63 bytes (or **504 bits**) of memory. 'Multi-color' Sprites reduced the horizontal resolution to 12 (double-wide) pixels and the same 21 vertical pixel resolution, however only two additional colors could be chosen and were shared across all 8 Sprites. Things are different now.

---

* the other thing I '*love*' about the code above is the line numbering: 1, 5, 10, 11, 12 ... COME ON, COMMODORE !!!

Foenix Systems support *full* color Sprites which are 32 pixels by 32 pixels and may use one of several 256-color, user definable palettes. From a BASIC816 perspective, each palette color is derived from a 24 bit RGB value, thus requiring 3 bytes per color * 256 colors or 768 bytes per shared palette.

Looked at through a 'binary pixel' lens, the bitmap portion of the Foenix scheme requires **1024 bits** (about twice that of the C64 due to it being 8 bits wider and 11 bits taller); but instead of storing only one bit ('on' or 'off') per pixel, VICKY II requires a byte per pixel and the value of the byte identifies a color number within the selected palette. In total, each Sprite will require **1024 bytes**. Color palettes, which are shared with bitmaps and tiles, are stored separately.

640 x 480 resolution - *not to scale* with the C64 example above

The following BASIC816 program, based on Peter Weingartner's GitHub hosted example, will perform the Sprite setup required for a 'dull-yellow-square[tm]' on the Foenix platform. Compare some of these commands to the methods used in the Commodore example above.

```
10    CLS : GRAPHICS &h27                                  : REM Sprites with text overlay
12    SETCOLOR 0, 1, 128, 128, 0                           : REM A dull yellow
16    FOR x% = 0 TO 1024 : POKE &hB00000 + x%, 1 : NEXT    : REM Define sprite image (just a yellow square)
18    SPRITE 0, 1, &hB00000                                : REM Set up sprite #0
20    SPRITESHOW 0, 1                                      : REM Make sprite 0 visible
22    SPRITEAT 0, 100, 100                                 : REM Position sprite 0 to 100, 100
```

**Related topics and resources**

Whether modern Foenix or ancient Commodore, there are 4 or 5 steps required to instantiate a Sprite (and potentially others) in order to get them to provide value in the context of a game. These topics might include:

- Non-linear movement
- Animation
- Collision detection (with other Sprites or Background text, Tiles, or Graphics)
- Binding animation or movement to an interrupt and controlling smooth movement via a time-slicing engine

The next issue of *Foenix Rising* will discuss at least one other use of Sprites; as an effective tool for non-movable objects and across the next few issues, we will delve into related topics.

For a more detailed video review of Foenix Graphics (in general) and BASIC816 commands, click here; and for an overview of Sprites in the context of 65816 Assembly language, you can find a nice video tutorial and others here. Both were produced by Peter Weingartner, the developer behind BASIC816 and the Kernels of various Foenix machines.

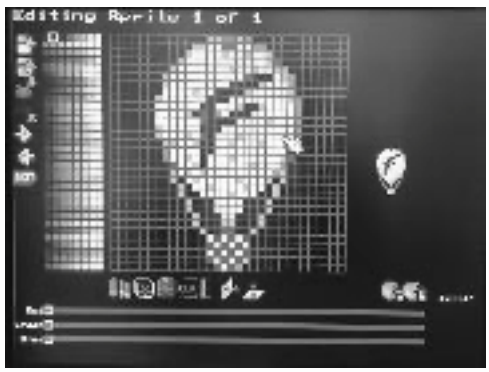
*Note: at least one of these videos was produced 2+ years ago (at the time of this writing) and VICKY II capabilities may have changed depending on when you are reading this.*

**Tooling for creating Sprites**

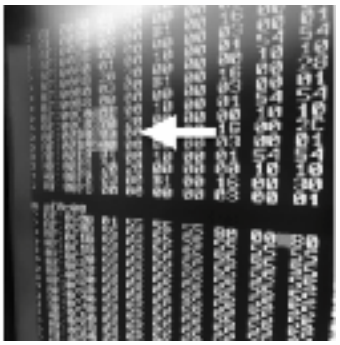
Prior to the availability of Sprite editors, the tool of choice was graph paper and binary encoding. In fact, Commodore, in their C64 User's Guide manual, included a grid depicting and instructing just that, in connection with the Sprite on page 8.

Thankfully, several generations of image editors have brought pixel-art to multiple platforms, and Foenix is no exception.

Ernesto Contreras released Foenix Sprite Editor in 2021 for BASIC816 (FMX and C256U) platforms. We will be doing a deep dive into this app in the next issue, but don't wait for it; get your hands on it now because: a) it takes the toil out of editing Sprites; and b) it's colorful and fun to use. (Look for it on the Foenix Marketplace towards late July 2022.)



*Foenix Sprite Editor* by Ernesto Contreras (used to create the Foenix Balloon)



**My first Foenix Sprite**: my first experience (and my first modest *success*) with the Foenix platform came shortly after powering up my C256U+ in October. Equipped with the built in ML monitor and a VICKY address register map (and **no** Sprite editor !!), I posted the pic that you see to the left.

The white arrow points to my very first Sprite; random bits tied to random colors (before I knew how Foenix platforms leveraged such features) but I could move it by typing hex values into registers. It resembled a piece of moldy cheese (aged, relative to Peter's yellow square).

The second Sprite I created was the Foenix Balloon above. I made the original version of this at 2am following day-one of VCF East. Even if you do not consider yourself a graphics artist, Google "pixel art Pacman" or for a Space Invaders alien and have at it.

**5 simple steps**

Let's move along to five steps to instantiate a *non*-cheese-like Sprite and then we will briefly discuss horizontal, vertical, and diagonal movement. Basic steps include:

1. Selecting graphics mode -> 2. Setup Sprite color(s) -> 3. Bind bitmap and LUT data to the Sprite -> 4. Identify coordinates -> 5. Make it visible

The prerequisite to the multi-color Foenix Balloon is loading the binary image file "`FBALLOON.SPR`" and establishing a color lookup table (embedded within data statements on the next page) into memory. We will cover the basics here and get into more complex subject matter on the next page.

Step 1: Select the graphics mode: **GRAPHICS**

- Select a **GRAPHICS** mode conducive to Sprite and Text; the BASIC816 command **GRAPHICS &h27** will select a combination of Sprite (bit 5) + Graphics (bit 2) + Text Overlay (bit 1) + and Text (bit 0) or 39 decimal aka 27 hex. See line #10 on pg. 11 or line #10 on pg. 9 to see this code in context.

Step 2: Define a palette aka look-up-table or 'LUT', if not already established: **SETCOLOR**

- **SETCOLOR** `<lut>, <color #>, <red>, <green>, <blue>` for each color used; you do not need to define all 255 colors, but you should define the colors required for your Sprite. Binding a Sprite to a LUT is accomplished in step 3. See line #20 and the called subroutine on pg. 11 for the full color palette example, or line #12 on pg. 9 above to see a simple example of how to use. It's subtle, but the use above defines but a single color (color #1) on LUT 0 using RGB value 128, 128, 0 (putrid). Sometimes, one or two colors is all you need.

Step 3: Bind the Sprite to a color palette/look up table and optionally, identify the VRAM where the Sprite bitmap data is stored

- **SPRITE** `<sprite #>, <lut> [, <address> ]` if the optional `address` is not identified, the Sprite will default to B00000. See lines #30-50 on pg. 11 where we load our Sprite into memory from disk, use the `memcopy` command (more on this next issue) to move it into video memory at location B00000, and then explicitly bind Sprite #0, using LUT #0 and notation &hB00000 with the **SPRITE** command.

Step 4: Position the Sprite with **SPRITEAT** (aka provide the x and y coordinates):

- **SPRITEAT** `<sprite #>, <x>, <y>` Sprite # from 0 .. 63, and coordinates from 0, 0 to a valid value which takes screen resolution into consideration. As you will see in the examples below, x = 0 and y = 0 is 'offscreen' (under the border).

Step 5: Enable or turn on the Sprite:

- **SPRITESHOW** `<sprite #>, <visible> [, <layer> ]` where `visible` may be set to 0 (hide) or 1 (visible); optionally, a layer number may be provided. In the simple example on pg. 9, Sprite 0 is positioned at 100, 100 (near the upper left corner). Below, we will move the Sprite using a few different methods.

**Color Palette Setup & Sprite Positioning / Movement**

The following BASIC816 code, borrowed from Ernesto's *Foenix Sprite Editor*, incorporates color data from the following resource: https://lospec.com/palette-list/duel. Ernesto carefully converted the color codes into data statements, modifying certain colors slightly, and added a four line BASIC loader.

RGB values are encoded into BASIC `DATA` statements from line 20000-20510 and a simple subroutine on lines 10020-10050 loads LUT 0 with data using the BASIC816 **SETCOLOR** command on line 10040.

One aspect of the VICKY II LUT that may not be obvious, is the first color in each table is *reserved* for transparent (aka, clear). As such, the first 3 values of data on line 20000 (`"0, 0, 0"`) are ignored. The second color in this case happens to also be (`"0, 0, 0"`), but is black; the last color, represented by the final 3 values in the table on line 20510 (`"255, 255, 255"`) is white. We could have just as easily had color 1 = white, color 2 = black and color 3 = red and not bothered with the rest, but our Sprite uses several colors and it's nice to just leverage the palette in the Sprite Editor.

In our next *Beginner's Corner*, we will be discussing palette cycling and dynamic color reassignment (two methods for animating that do not require additional Sprite definitions).

```
10020 FOR c%=0 TO 255 : REM load palette
10030 READ r%,g%,b%
10040 SETCOLOR 0,c%,r%,g%,b%
10050 NEXT : RETURN

20000 DATA 0,0,0,0,0,0,34,35,35,67,69,73,98,104,113,130,139,152
20010 DATA 166,174,186,200,200,200,98,93,84,133,117,101
20020 DATA 158,140,121,174,161,137,187,175,164,204,195,177,234,219,201
20030 DATA 255,243,214,088,49,38,115,61,59,136,80,65,154,98,76
20040 DATA 173,110,81,213,141,107,251,170,132,255,206,127,00,39,53
20050 DATA 0,56,80,0,77,94,11,102,127,0,111,137,50,140,167
20060 DATA 36,174,214,136,214,255,102,43,41,148,54,58,182,77,70
20070 DATA 205,94,70,227,120,64,249,155,78,255,188,78,255,233,73
20080 DATA 40,43,74,58,69,104,97,95,132,122,119,153,134,144,178
20090 DATA 150,178,217,199,214,255,198,236,255,000,34,25,0,50,33
20100 DATA 23,74,27,34,89,24,47,105,12,81,136,34,125,164,45
20110 DATA 166,204,52,024,31,47,35,50,77,37,70,107,54,107,138
20120 DATA 049,142,184,65,178,227,82,210,255,116,245,253,26,51,44
20130 DATA 47,63,56,56,81,64,50,92,64,65,116,85,73,137,96
20140 DATA 85,182,125,145,218,161,94,7,17,130,33,29,182,60,53
20150 DATA 228,92,95,255,118,118,255,155,168,255,187,199,255,219,255
20160 DATA 45,49,54,72,71,77,91,92,105,115,115,127,132,135,149
20170 DATA 171,174,190,186,199,219,235,240,246,59,48,60,90,60,69
20180 DATA 138,82,88,174,107,96,199,130,108,216,159,117,236,197,129
20190 DATA 255,250,171,49,34,42,74,53,60,94,70,70,114,90,81
20200 DATA 126,108,84,158,138,110,192,165,136,221,191,154,46,16,38
20210 DATA 73,40,61,102,54,89,151,84,117,185,109,145,193,120,170
20220 DATA 219,153,191,248,198,218,0,46,73,0,64,81,0,81,98
20230 DATA 0,107,109,0,130,121,0,160,135,0,191,163,0,222,218
20240 DATA 69,49,37,97,74,60,126,97,68,153,121,81,178,144,98
20250 DATA 204,169,110,232,203,130,251,234,163,95,9,38,110,36,52
20260 DATA 144,70,71,167,96,87,189,125,100,206,151,112,237,182,124
20270 DATA 237,212,147,50,53,88,74,82,128,100,101,157,120,119,193
20280 DATA 142,140,226,156,155,239,184,174,255,220,212,255,67,23,41
20290 DATA 113,43,59,159,59,82,217,74,105,248,93,128,255,125,175
20300 DATA 255,166,197,255,205,255,73,37,28,99,52,50,124,75,71
20310 DATA 152,89,90,172,111,110,193,126,122,210,141,122,229,154,124
20320 DATA 032,41,0,047,79,8,73,93,0,97,115,8,124,131,30
20330 DATA 150,154,38,180,170,51,208,204,50,98,42,0,117,59,9
20340 DATA 133,79,18,158,101,32,186,136,46,209,170,57,232,210,75
20350 DATA 255,246,79,38,35,61,59,56,85,86,80,111,117,104,110
20360 DATA 145,122,123,179,151,131,207,175,142,254,223,177,29,44,67
20370 DATA 46,61,71,57,77,60,76,95,51,88,113,44,107,132,45
20380 DATA 120,158,36,127,189,57,55,36,35,83,57,58,120,76,73
20390 DATA 148,93,79,169,109,88,191,126,99,215,147,116,244,163,128
20400 DATA 45,75,71,71,101,90,91,123,105,113,149,125,135,174,142
20410 DATA 138,193,150,169,209,193,224,250,235,0,27,64,3,49,95
20420 DATA 7,72,124,16,93,162,20,118,192,64,151,234,85,177,241
20430 DATA 109,204,255,85,71,105,118,93,115,151,116,136,185,140,147
20440 DATA 213,163,154,235,189,157,255,213,155,253,247,134,29,29,33
20450 DATA 60,49,81,88,74,127,121,100,186,149,133,241,169,150,236
20460 DATA 186,171,247,209,189,254,38,36,80,40,51,93,45,61,114
20470 DATA 61,80,131,81,101,174,82,116,197,108,130,196,131,147,195
20480 DATA 73,33,41,94,65,74,119,83,91,145,96,106,173,121,132
20490 DATA 181,139,148,212,174,170,255,226,207,114,28,3,156,51,39
20500 DATA 191,90,62,233,134,39,255,177,8,255,207,5,255,240,43
20510 DATA 255,255,255
```

The following example represents the baseline setup, required for each of the examples below (note that line 20 calls the Color Palette setup sub-routine to the left).

```
10    CLS : GRAPHICS &h27
20    GOSUB 10020
30    BLOAD "FBALLOON.SPR",&h100000
40    MEMCOPY LINEAR &h100001,1024 TO
      LINEAR &hB00000,1024
50    SPRITE 0, 0, &hB00000 : SPRITESHOW 0, 1
60    SPRITEAT 0, 100, 100 : END
```

**BLOAD, MEMCOPY,** and *memory,* will be discussed next issue

Don't forget to *show* the Sprite

**END** is required here to prevent this code from running into line 10020

Stationary balloon placed at 100, 100

The following example demonstrates x-axis movement (an endless loop that increments the `x%` variable from location 0 to location 200, leaving the y-axis static @ 100).

```
60    FOR X% = 0 TO 200 : SPRITEAT 0, X%, 100 : NEXT : GOTO 60
```

X-axis movement; Y-axis fixed @ 100

These examples are endless loop; press **BREAK** to stop them

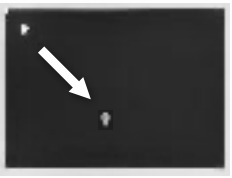
Next, we move vertically by incrementing `y%`, with a static x-axis @ 100.

```
60    FOR Y% = 0 TO 200 : SPRITEAT 0, 100, Y% : NEXT : GOTO 60
```

Y-axis movement; X-axis fixed @ 100

This example increments `n%` applied to x-axis and y-axis, resulting in diagonal movement on a pixel by pixel basis

```
60      FOR N% = 0 TO 200 : SPRITEAT 0, N%, N% : NEXT : GOTO 60
```

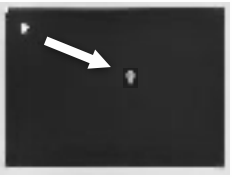


Diagonal movement;
X and Y increment identically

If you find that these run too fast for your liking, insert a delay loop; we will cover this next issue

And finally, this piece of code increments `y%` once for every two `x%` increments, resulting in something closer to a 22.5' (relative to horizontal) angle. It's not a precise 22.5' angle due to the fact that pixels are not perfectly square. We are leveraging the BASIC modulus operator to test for even values, and are also using a variable called `MAX%` to identify the maximum x-axis value. We will use this next time.

```
60      X% = 0 : Y% = 0 : MAX% = 200
70      SPRITEAT 0, X%, Y% : IF X% = MAX% THEN 60
80      X% = X% + 1 : IF X% MOD 2 = 0 THEN 100
90      GOTO 70
100     Y% = Y% + 1 : GOTO 70
```



A more shallow 2:1 angle;
X and Y incrementing based on an operator and a decision statement

---

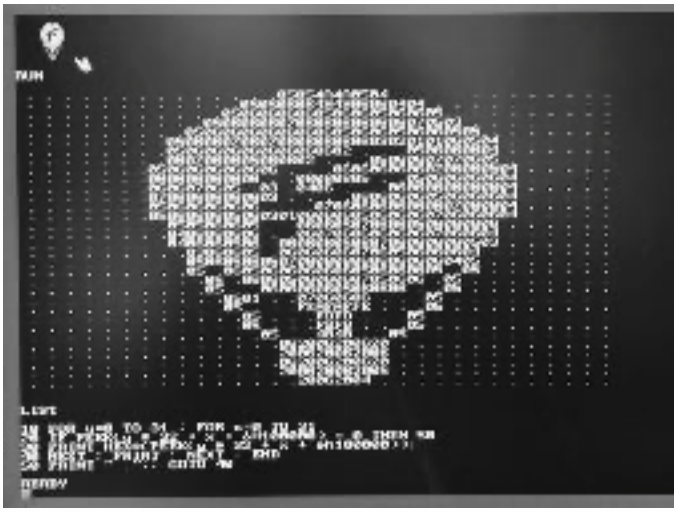
**Next time …**

The 'tool' depicted in the graphic on the right (a five line BASIC816 program) will be instrumental in our next steps with this project.

A significant part of software development is becoming familiar with a given "tool chain" and sometimes, you just have to create your own tools and methods to manage and interrogate data.

We could use Gimp (GNU Image Manipulation) and a set of Python scripts on other platforms and sneaker-net files back and forth, but let's stay here if we can. That's what younger versions of ourselves had to do.

For lightweight work, I think it's the right answer and we will benefit by doing more with these new platforms rather than getting distracted by others.



The aim for this column is not only to provide insight into at least 'one' way to leverage Foenix features, but also to challenge and experiment, and in-turn, share findings, code examples, and applications & amusements with the community. Many are already doing this on the Discord hosted Foenix Channels. If you have not yet joined, you should do so via the 'invite' link on pg. 2 above.

# Back Page - Vintage Advert Time Machine

Skyles Electric Works “1541 Flash!” - developed by Bryce Nesbitt

Many of us remember the *Epyx Fastload* cartridge (I still have mine). In the mid-80s, there were dozens of disk speedup products, and in fact, an entire ecosystem built around products for and about the Commodore family of disk drives existed.

The speed of the 1541 was legendarily slow but the complexity was impressive and capabilities surprising. The 1541 (and the 2031, 4040, 8050, 8250, and later the 1551, 1571, and 3.5” 1581) was a powerful, autonomous device housing its own 6502 processor, ROM, RAM, and an operating system developed to communicate with the attached Commodore computer and in some cases, with other drives in peer to peer fashion. They could also do this, and more recently this!

But the 1541, as shipped, was slow as molasses thanks to a serial protocol and a hardware issue that dated back to the 1540 drive in the VIC20 days. Some of the speed issues were dealt with as newer Commodore computers began adopting the older (PET IEEE) standard; this included the higher density SFD-1001 drive (yea, I owned one of those too), or in the case of the TED series machines, the 1551 drive.

Sometime between 1984 and 1985, I traversed the hills of Berkeley California to visit the Berkeley Commodore Computer club which met at the Lawrence Hall of Science. I have no recollection of how I learned about this particular meeting; it must have been posted on a local BBS.

But my intention was clear; Skyles Electric Works would be demoing and selling the *1541 Flash!* product and I had to check it out!

The ad to the right, as awesome as it is, does not portray the full picture; 1541 Flash! was part hardware and part embedded (ROM based) software. Unlike other SW only solutions, this product included wiring that ran parallel to the IEC (serial) cable offering a parallel data path. It also included custom ROMs for the 1541 and the C64, and SW and HW switching to disable the product. Many of the disk speedup products of the day got in the way of some of the copy protection schemes, or occupied one or more ports or addresses, otherwise needed for something else.

1541 Flash! had other features such as an embedded DOS wedge but it was not as fast as the EPYX product, took a bit of work to install, and was somewhat expensive. Have a read of the doc here.

**1541 FLASH!™— THE CRITICS SPEAK OUT**

This advertisement has been written by independent reviewers of Commodore computer products. We thank them for their frankness and the high marks awarded to the Skyles Electric Works **1541 FLASH!**.

**The Reviewers Comment**

“Tired of that slow disk drive? The solution is here, and it costs about $90. Considering what it does, the price is a bargain! It speeds up your 1541 disk drive 200%–300%. And if you write your own software specially for the **FLASH!**, you can achieve speed increases of 600%!

The **1541 FLASH!** is the best! It's better than KWIK LOAD.™ And better than 1541 Express.™

Does it sound too good to be true? Do you suspect there must be some drawback that I haven't mentioned? Well…There is one. You have to open up your keyboard and 1541 drive and do a little work inside them. You need to replace a couple of chips with new ones provided by the **FLASH!**. And an extra cable will run from your keyboard's user port to your 1541 drive. But the installation is explained in complete detail with pictures. It's a simple operation that will take under 30 minutes. And in return you will have a disk drive that literally races along!

The biggest complaint with the Commodore 64 is the slow disk drive. No more! You will never be willing to go back after having used the **1541 FLASH!**. It really spoils you! It's even faster than a parallel drive with an IEEE interface! Don't be afraid of the installation. It's really simple. And if you prefer not to do it yourself, your local user's group probably has people with the ability to install it for you. You'll be glad you did!”

*The Northwest Users Guide, Jan. 1985*

“A tiny wedge is included…you simply SYS65526 to enable it. Those who enjoy using the wedge as part of their normal computer will like this feature.

The utilities added by **FLASH!** include single, double and simultaneous keystroke implementation of such niceties as delete line, escape quote, cursor to bottom of screen, 16 character tab, and return without line execution.

My children have played some of their favorite games and used utilities or educational software without any problems whatsoever, using the **1541 FLASH!** All in all, this device will save the purchaser much of the frustration normally experienced, as well as the time required in normal disk drive-computer interactions. With the above noted exception, I'm very pleased with its operation, and won't hesitate to recommend it to those who would like faster loads and saves or want additional flexibility and power at a moderate price.”

*RUN, May 1985*

“Having used **1541 FLASH!** for several days, already I can't bear to go back to a slow 1541. It's amazing how quickly you can get spoiled by a luxury like this. More than a few editors here have cast covetous eyes on the upgraded 64/1541. And the price is reasonable for such a dramatic enhancement. After reading the installation description above, you can decide if you're up to opening your equipment, yanking out chips, and enjoying a 300% speedup.”

*Compute!'s Gazette, Jan. 1985*

**Commercial Details**

| | |
|---|---|
| **1541 FLASH!** | $ 89.95* |
| **1541 FLASH!** with Disk Switchboard | 99.00* |
| **1541 FLASH!** for Two Drives | 139.95* |
| **1541 FLASH!** for SX-64 | 99.95* |
| **1541 FLASH!** for MSD Drives | tba* |

*Mail order please add $3.50 for shipping and handling. California residents add sales tax as required.

**Available from your local Commodore dealer or call 1-800-227-9998**

**Skyles Electric Works**
231E South Whisman Road
Mountain View, CA 94041
1-(415) 965-1735

1541 FLASH! is a trademark of Skyles Electric Works. Commodore 64 is a trademark of Commodore.

Reader Service No. 188

I was lucky enough to have met the developer of the product, Bryce Nesbitt, that evening. As it turned out, he was local to me (I lived just off campus on Walnut Street and he lived about a mile away); he offered to install it for me, later giving me a custom ROM burned with my choice of screen colors; what a guy!

Bob Skyles, the owner (a former Commodore employee) reminded me of Colonial Mustard and had a nose for solutions for Commodore products. He began by selling ‘external’ clunky replacement keyboards for the PET but in the end, he couldn’t keep up with software-only solutions that ended up dominating the industry; but he had a good run.

Bryce went on to work for Commodore and specifically, Amiga. You can learn a bit more about him from the Viva Amiga documentary outtakes. He was a super sharp engineer and a genuine and generous guy. This little known product, while niche, was an amazing technical accomplishment, nearly lost in the annals of time.