Foenix Rising

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

Issue #3 - update 1 September / October 2022

Early Arcade issue

This issue pays homage to *early* coin-op arcade. The initial intent was to start with the history of Atari *Tank*, but after some research, we've opted to take a step even further back (6 more months) to Atari's May 1974 release of "Gran Trak 10".

Gran Trak was an early, if not the earliest driving game and was released 18 months after Nolan Bushnell's first foray, *Pong*. It is also known for almost pushing Atari to insolvency.

Tying this to August's "Classic Arcade" puzzle, we will focus on a set of titles from 1980 to 1984 through the lens of my experience (and maybe yours), being a teenager in the '80s.

The 2nd half of this issue is all Foenix and we will begin by challenging you with a puzzle of a different sort to test your familiarity with hardware component use in a "what's wrong with this picture" activity.

Ernesto Contreras is back as a contributing developer with an article on the Foenix RTC (real time clock). We will also cover the v1.1 update to his Foenix Sprite Editor and discuss how to leverage color palettes saved from the editor.

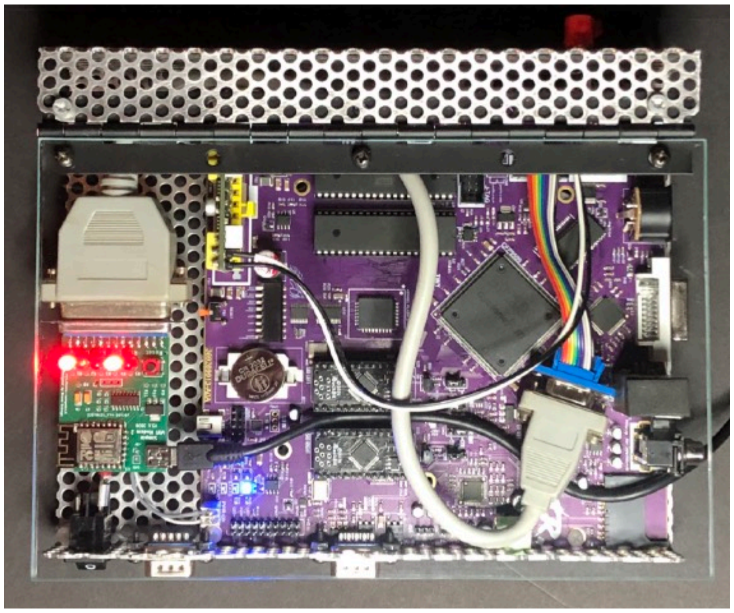
Beginner's Corner closes our 3-part Foenix Balloon series by throwing as much variation as possible into a 25 line BASIC816 program.

Next issue, we will focus on A2560K. Mine has been neglected, and is sitting on my piano looking angry!

Thank you, as always.

-EMwhite

Featured Photo - 'wired and ready' - C256 Jr. dev board



Last month we introduced the C256 Jr. by way of a Rev A dev board that arrived just prior to going to 'press'.

This month, we discuss ITX power, and share a few action shots of my one-off concept case.

But the *big news* is, the prod candidate Rev B board was submitted for fabrication earlier this week. Renamed, the **F256 Jr.**, there is even more utility packed into the small package. Pre-order is now open.

VTOC - volume table of contents

Resources, publisher's notice, and Arcade word search solved	2
Gran Trak 10 - Atari's 1974 'driver' - the first of many to come	3-5
Classic Arcade - We discuss late '70s and early '80s Arcade and examine a handful of noteworthy titles released between 1980 - 1984	6-12
Arcade before Arcade - Sega's wonderful mid '60s and early '70s Electro-mechanical Arcade in pictures	13
Pacman - An examination of an interesting aspect of the original design & a preview of a Ms. Pacman demo, actively being ported to Foenix C256	14-15
"Just 4 Fun" - A 'tear-out' activity book exercise for the kid in all of us. *MAD Magazine* meets *Highlights Children's Magazine* meets Foenix	16-17
Date and Time on your Foenix - written by Ernesto Contreras	18-20
Foenix Sprite Editor update: Ernesto releases v1.1 and we discuss some of the new features, plus, saving and leveraging palette LUTs	21-23
Beginner's Corner #3: Lightweight Sprite animation in 25 lines or less !! Revisiting last month's challenge with a complete & working prototype	24-28
Foenix Junior - The Prod release is officially announced and available on pre-order, rebranded as the "F256 Jr."	29-32

`git` and URL Resource Directory

Updated each issue, this space contains links to public Foenix related development efforts

Lang	https://github.com/daschewie/FoenixBasic68k
Utility	https://github.com/daschewie/FoenixEdit
Game	https://github.com/dtremblay/c256-tetris
Utility	https://github.com/dtremblay/c256-vgm-player
Game	https://github.com/dtremblay/fraggy
Utility	**https://github.com/econtrerasd/Foenix-Sprite-Editor**
Utility	https://github.com/econtrerasd/playSong
Library	https://github.com/econtrerasd/VickyGraph
Kernel	**https://github.com/ghackwrench/OpenKERNAL**
Utility	https://github.com/hth313/Calypsi-Foenix-guide
Utility	https://github.com/hth313/petit-fatfs-foenix-jr
Samp code	https://github.com/noyen1973/C256-Foenix
Utility	**https://github.com/paulscottrobson/junior-utilities**
Lang	**https://github.com/paulscottrobson/superbasic**
Env	https://github.com/Trinity-11/FoenixIDE
Utility	https://github.com/vinz6751/FoenixSamples
Env	**https://github.com/vinz6751/genxtos**

highlighted = mentioned this issue **bold** = newly added or updated

Links to other Foenix Resources:

Foenix Retro Systems Home Page
Foenix Discord Invite
Stefany Allaire Patreon Page
Stefany Allaire Twitter
Foenix Marketplace content 'store'
VCF East 2022 Foenix Booth (virtual tour)

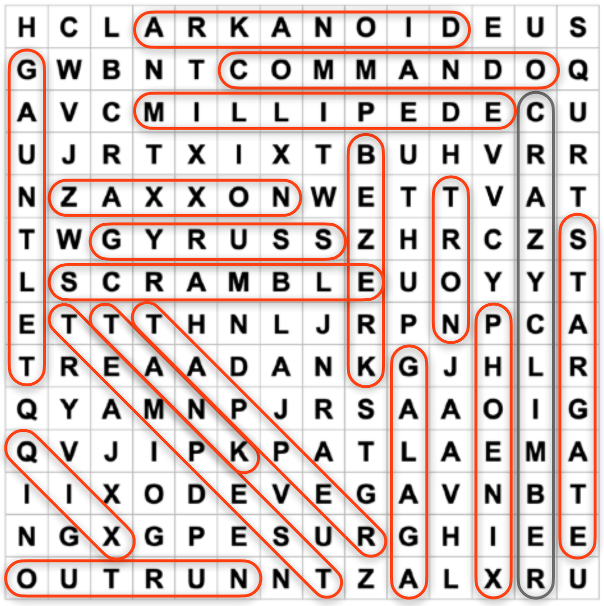
Foenix Rising is a user-supported, not-for-profit bimonthly hobbyist's newsletter published in Murray Hill, New Jersey, USA supporting Foenix Retro Systems products with a focus on software development and related retro technologies.

Distribution: ~2^{10}-1

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

Correspondance: `commerce@emwhite.org`

Classic Arcade Games



H	C	L	A	R	K	A	N	O	I	D	E	U	S
G	W	B	N	T	C	O	M	M	A	N	D	O	Q
A	V	C	M	I	L	L	I	P	E	D	E	C	U
U	J	R	T	X	I	X	T	B	U	H	V	R	R
N	Z	A	X	X	O	N	W	E	T	T	V	A	T
T	W	G	Y	R	U	S	S	Z	H	R	C	Z	S
L	S	C	R	A	M	B	L	E	U	O	Y	Y	T
E	T	T	T	H	N	L	J	R	P	N	P	C	A
T	R	E	A	A	D	A	N	K	G	J	H	L	R
Q	Y	A	M	N	P	J	R	S	A	A	O	I	G
Q	V	J	I	P	K	P	A	T	L	A	E	M	A
I	I	X	O	D	E	V	E	G	A	V	N	B	T
N	G	X	G	P	E	S	U	R	G	H	I	E	E
O	U	T	R	U	N	N	T	Z	A	L	X	R	U

Across a period of only 5 to 8 years, coin-op solid-state and microprocessor based Video games overtook Pinball's 30+ year head start, ushering in modern incarnations of the Electro-mechanical Arcade that once populated the midway.

Pinball benefitted from the advent of solid-state electronics as well, but the [then] modern Arcade (which we now call 'classic') kick-started a craze that swept across Japan, North America, and the world.

The earliest mechanical Arcade games gave designers a blank canvas, limited only by the imagination of design teams. In reality, material manufacturing limits, physics, and cost played a role.

Over time, leaf switches, motor driven belts, stepper and score motors/reels, and plastic moulded scenery was displaced by a 13" CRT, one or more general purpose microprocessors, and numerous integrated circuits. The result was otherworldly. Imagine trying to implement Pacman, Robotron 2084, or Marble Madness in the physical world. Attempting such a feat would be, well… madness!

40 years later, it would not be incorrect to suggest that the advent of Arcade and the tech that drove it sparked a fire that attracted an entire generation of young people to a then 'new' curriculum called "Computer Science". It also introduced gaming consoles and home computers to households.

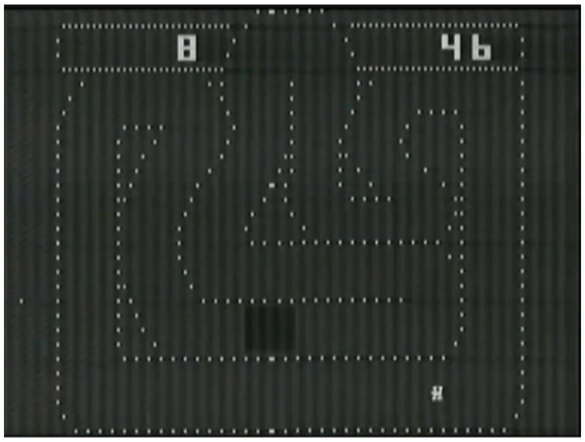
With a few exceptions, the titles above were among the most popular and innovative. The front 1/2 of this issue of Foenix Rising is dedicated to coin-op Arcade with Foenix tie-ins sprinkled throughout.

Atari Gran Trak 10

Examining Atari's first driver and the genre that evolved from it

The first driving game

Gran Trak 10 is widely acknowledged as the first CRT based driving game, establishing the genre almost 50 years ago. You can graph a straight line from Atari's Gran Trak and *Sprint* titles to *Night Driver* and ultimately to *Pole Position* (which led the way for non-Atari titles such as OutRun and Crazy Taxi).



In truth, the Atari lineage bounced back and forth and around, between overhead, top-down, and 'first-person' drivers, many times; who can forget Atari *Fire Truck*? That relentless siren is hard to forget!

Gran Trak is said to have been influenced by Nolan Bushnell's experience working in electro-mechanical arcade during college and exposure to Chicago Coin's Speedway, specifically.

In this article, we will walk through the history including aspects of architecture from this influential starting point. We will close with a prototype idea for a Foenix platform game (based on actual Gran Trak 10 assets).

Brief History and Highlights of Gran Trak 10

In the early '70s, Atari partnered with Cyan Engineering, located in Grass Valley, California. Cyan became instrumental in the development of the 2600 (then called the VCS) and Atari 400/800 8-bit computer architectures; Larry Emmons (one of Cyan's founders) is given credit for creating Gran Trak 10.

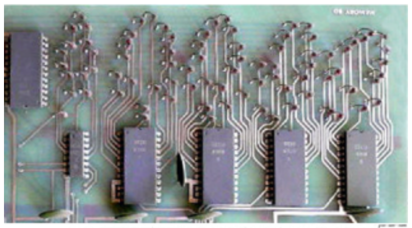
Larry Emmons can be seen in this documentary video recorded in 1982 (transferred from VHS). It's a little rough on the eyes. Also, the video quality is poor. That's Larry sitting at the roll-top desk @ 1:05. His laid back demeanor reminds me of actor John C. Reilly, but with a mustache.

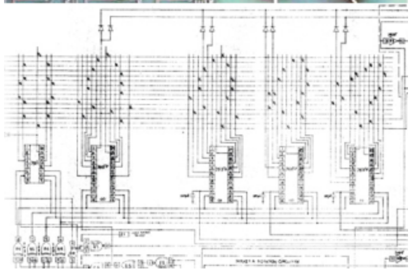
Emmons and Atari found that interfacing physical controls with solid state electronics was not an easy task; sourcing components and contracting fabricators is tricky and expensive, and it nearly cost Atari as they slipped close to bankruptcy (they were reportedly losing $100 per game and operating at a $500K deficit before being acquired by Warner Communications in mid-1976).

According to internet sources, it took a few iterations to get the product to work well; remember, this was presumably *the* first use of a steering wheel, a gas pedal, brake pedal, and a four-position stick shift.

This was also an early use of a rudimentary ROM to house graphics data (for the car and a track layout). Prior efforts dating back to the famous *Computer Space* game used on-board diodes which were expensive and took up a ridiculous amount of circuit board real estate.

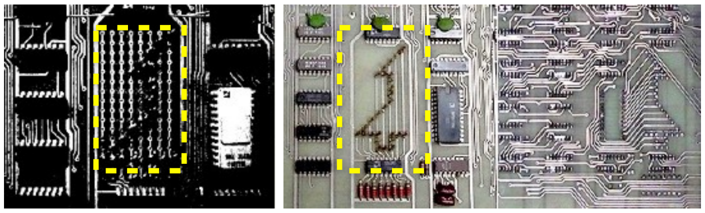
Here are two examples of this early use. *Computer Space* 1971 by Nutting Associates (on the left) and Atari's *Space Race* (the followup after Pong) produced in 1973, below.

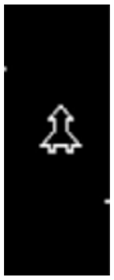




For some unknown reason, designers opted to fashion diodes in the shape of the actual on-screen object (possibly to aid in trouble-shooting should a pixel/data bit go dark). Good documentation, I suppose.

Footnote on the Space Race example below; the dark pic on the left contains a prototyping area/peg board, presumably for 'editing' the first sprite. Do you hear that, Ernesto!? And I thought you invented the concept.





Ah, the roots of Atari's VCS screen mirroring scheme; when 'memory' was scarce and expensive, mirroring was leveraged. Necessity, the mother of invention, indeed.

The graphic to the left was captured from a YouTube video demonstrating the early Atari Space Race game, said to be the first "racing" game. Arcade has come a long way.

Technology - Discreet Logic rules the day

Gran Trak 10 belongs to a distinctive class of arcade built from discreet logic ICs. In other words, it is not microprocessor controlled and therefore, has no code or software of any kind. This title is somewhat unique since it contains a ROM (of sorts), and was reportedly, the first to do so. It is also the first to connect a steering wheel to solid state for the purpose of simulating driving.

From a technical perspective, Gran Trak more closely resembles a state machine with different sections of board logic managing 'state' of an element of the game. As a quick example, the steering wheel is 'wired' to the car image ROM select regardless of where the car is on the screen or what might be going on in an "interrupt loop". Not unlike electro-mechanical pinball amusement or arcade, flip-flops (though solid state) are aplenty, except in this case, they are packaged on TTL ICs rather than dual-state relays with physical latching mechs.

If you've followed Ben Eater's tutorial series, you are probably aware of his "8-bit Computer" project. Across several hours of video, Ben provides a tutorial to literally build a microprocessor (or a rather *large* processor) using nothing but TTL ICs and wire.

Impressively, Ben implements an ALU (arithmetic logic units), he reads microcode/data in order to fashion control-words from ROM, and stores data (RAM). He also constructs a program counter, and negotiates chip selects in support his "instruction set".

Others have done wondrous things with similar design tenets (discreet logic only) building fully functioning computers complete with color and graphics. See Gigatron for once such example.

The point is, Gran Trak 10 is somewhere in the middle (from a year of release perspective) of a group of games that did *not* use CPUs. Most impressive to me is the video technology that became increasingly complex all the way up to the **pinnacle** of discreet logic video arcade (also a driver); Gremlin/Sega's Monaco GP. (if you are ever in Las Vegas, Tim Arnold has one on coin-drop)

There is plenty to be said about the analog sound effects and the interlaced video modes (2 x 260.5 horizontal scan lines) used in Gran Trak and others like it. More stories for another time.

Alternate Track(s)

Gran Trak 10 had but a single track, but some owners have modded their machines to access the 2nd, *hidden* track. The mod requires grounding (or alternatively, applying +5V) to a pin on one of the quad NAND gates (a DM7400N adjacent to the track ROM).

The alternate track was leveraged when Gran Trak was sold as *Formula K* through Atari's "Kee Games" division, but you can have both with this mod. This link contains all of the graphics data for the track including the digits for score and time, plus the animated frames of the car.

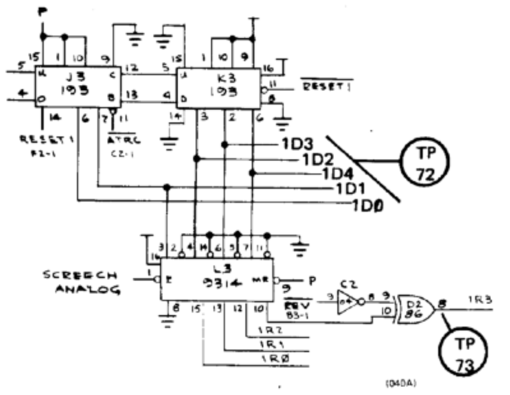
Documentation

The Gran Trak documentation may possibly be the best manual ever written (not counting my Volvo P1800 repair manual set). In all seriousness, typesetting in the mid-'70s was an art, especially when detailed diagrams and pictures were included. Outside of what Steve Wozniak did with the early Apple products, for one reason or another, arcade and computer manuals slipped over the years and are now non-existent.

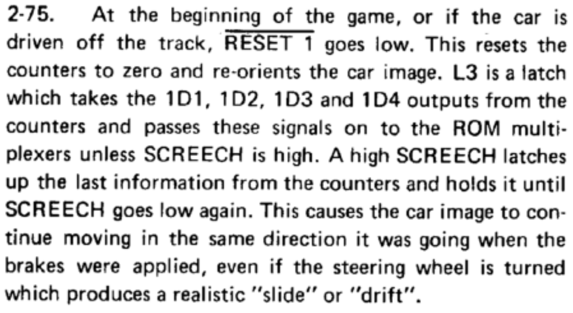
In the Gran Trak manual, there is a blend of theory of operation and test/troubleshooting information. Solid state circuits were relatively new in the '60s and '70s and thus, quality control and effects of heat had not yet reached the levels of reliability that we enjoy today. Maybe that was the driver (see what I did there?).

But most-most interesting, is the manner in which the manual explains circuit theory and troubleshooting procedures that you will never see for a piece of code that may, at some point, fail (contain a bug or stop working if a control signal or an actual IC failed). Remember, even though Space Invaders had an Intel 8080 and ROMs containing code, that didn't mean it did not have any other integrated circuits or passive components. Quite the opposite, the 3 board Space Invaders set was packed with logic, RAM and other ICs.

Here is example "2-75" of implementation detail from the Gran Trak manual. This is example #75 of 126. There are dozens of accompanying diagrams like this one:



For reference, 193's are binary up/down counters with 'clear' and the 9314 is a quad latch.


2-75. At the beginning of the game, or if the car is driven off the track, $\overline{\text{RESET 1}}$ goes low. This resets the counters to zero and re-orients the car image. **L3** is a latch which takes the **1D1**, **1D2**, **1D3** and **1D4** outputs from the counters and passes these signals on to the **ROM** multiplexers unless **SCREECH** is high. A high **SCREECH** latches up the last information from the counters and holds it until **SCREECH** goes low again. This causes the car image to continue moving in the same direction it was going when the brakes were applied, even if the steering wheel is turned which produces a realistic "slide" or "drift".

Other resources

Ed Fries published an outstanding repair blog, including dozens of links and numerous citations. He also interviewed the original Gran Trak engineers.

Gran Trak 10 Gameplay

With the exception of braking for turns, what you see is what you should expect in this game. Gran Trak is novel because it was the first, and is interesting because of its place in Atari's history. It is an extremely difficult game.

If you watch the handful of YouTube videos featuring this game, you'll find that owners are not particularly good at it, so there is great pride in succeeding to beat your prior score by a point or two in the quest to achieve the elusive, perfect race.

Driving titles from here:

An incomplete list of popular arcade driving games released between 1974 and 1986*.

Gran Trak 10 / 20 - Atari	Overhead	Solid State	1974
Speed Race - Taito	Top down vertical scrolling	Solid State	1974
Hi-Way - Atari	Top down scrolling	Solid State	1975
Sprint 2 - Kee Games	Overhead	Solid State	1976
Night Driver - Atari	1st person	Solid State	1976
Sprint 4 - Atari	Overhead	Solid State	1977
Super Bug - Atari	Top down omni scrolling	CPU	1977
Super Speed Race - Taito	Top down vertical	CPU	1977
Sprint 8 - Atari	Overhead	CPU	1978
Fire Truck - Atari	Top down scrolling	CPU	1978
Speed Freak - Vectorbeam	1st person	CPU	1979
Monaco GP - Gremlin/Sega	Top down vertical	Solid State	1980
Pro Monaco GP - Sega	Top down vertical	Solid State	1980
Rally X - Namco	Top down omni scrolling	CPU	1980
Turbo - Sega	1st person removed	CPU	1981
Bump 'n' Jump - Data East	Top down vertical	CPU	1982
Pole Position - Namco	1st person removed	CPU	1982
Spy Hunter - Bally/Midway	Top down vertical	CPU	1983
Pole Position II - Namco	1st person removed	CPU	1983
*Super Sprint - Atari	Overhead	CPU	1986
*Out Run - Sega	1st person removed	CPU	1986

* despite 1986 being beyond the reach of Warner Communications Atari (1985), I thought it important to highlight the *Super Sprint* title because it was a relative of the Atari Gran Trak / Sprint lineage. Also, Atari's System 2 was under development at the time of the acquisition by Namco (see the Marble Madness/System 1/System 2 section below for more on this).

Sega *Out Run*, also released in '86, was a Pole Position on steroids and led the way to dozens of games like it across home and arcade platforms. The post crash period still produced games, but abandoned the creativity of the prior period. For a fair 15 years, it felt as if every game was an also ran driving game, a fighting game, or a dance-dance whack-a-mole and Chuck-e-Cheese redemption game. Perhaps that was Nolan and Cyan's secret plan? (The same arm that produced Gran Trak developed the Chuck-e-Cheese robotics. Read about it here.)

Gran Trak Foenix - a proposal

This column within a column (in a single column) proposes a conceptual 'port' of Gran Trak for Foenix platforms.

The intent is to mirror, as closely as possible, the graphics, audio, and experience of the original game.

Graphics to be based on the original ROM images as linked on page 4 above including the block score and time numerals. These will be included in a `.SPR` file alongside car and track data. To support (include) the Junior, a 320 x 240 resolution will be the target.

Audio is TBD. The plan is to leverage the lowest common denominator of capabilities across the widest array of platforms. Ideally, this will be an FPGA based decision. Since the Junior is a work in progress, it is too early to tell.

Controller support will be limited to keyboard and joystick to start, but we may be able to integrate a 3rd party arcade spinner. At least one has a steering wheel add-on and similar to the original and leverages an encoding scheme that signals right or left movement (there is no notion of 12 o'clock or a 'home' orientation).

Track selection will be limited to the original Gran Trak 10, but longer term I'd like to include a superset of Gran Trak 10 (including Formula 'K', the Kee Games derivation), and Gran Trak 20. Ultimately, I would like to see somebody (you know who you are) develop a track editor but I have a tactical plan to address this "on platform". (See 'track encoding')

Extras - I said "model as closely as possible", but I would like to include excerpts from the original manual and promotional flyers within, memory permitting.

A note on track encoding - If 'tiles' are a collection of 8 x 8 or 16 x 16 images intended to construct a background (fixed or scrolling) and if sprites are typically a collection of 32 x 32 pixel objects intended for movable objects, what is the best way to store and optimize track data? Ideally it would be something that we can easily edit with existing tools without requiring the use of a Mac or PC to write python code.

I'm thinking of leveraging the Foenix Sprite Editor for unique 32 x 32 *super*-tiles, and a simple map file that dictates how super-tiles are arranged; finally an algorithm to unpack them such that the pylon / cone paradigm is preserved and car detection accomplished.

Timeline. Looking to have a working demo complete by mid-holiday break. As mentioned in "next for …" on pg. 28 below, after issue #4, I'll be moving to a quarterly format in order to allow myself more time for development and design and I'm hopeful that by the one-year mark of this publication that we have enough reader/developer contributions to keep this journal alive.

Feedback? - I'd be interested in hearing your thoughts and ideas. You know where to find me.

Classic Arcade

A look at the state of early gaming and an examination of five diverse examples

"Space Invaders, Asteroids, or Pacman?"

Ask an avid collector, vintage gamer, or retro aficionado, and you'll get an earful. There is 'first', most innovative, popular, cherished or rare; so many ways to measure and hundreds of great games released across just a few years; better clear your calendar for this one.

Slowly at first, then in rapid succession, Arcade games were released, gobbled up by operators, placed on location and played, encompassing tech innovation that was being developed in parallel. The result is a living body of history that many of us revel in.

Emulations, emulated emulators, battery powered handhelds, 60 in 1 JAMMA boards, cabinets, console Arcade Treasure collections, UltraCade, and countless tributes continue to surface; but let's talk about the technology, the history, and give a think about what might be possible on Foenix platforms. Facing facts, we didn't invest in 'home' computing to calculate small business payroll and catalog recipes, even though 'home' computing started that way.

In the beginning

The first games focused on sport, various types of racing, and warfare, of course. This is where the '60s and early '70s electro-mechanical (EM) arcade left off. Browsing the Sega EM titles across this period included:

- Periscope (submarine shooter)
- Moto Champ (motocycle)
- Sand Buggy (driving)
- Soccer (also sold as 'Mini-Futbol')

see pg. 13 for a photo gallery of this era of gaming

But a closer look at the Sega catalog (expertly curated at this tribute site) details a few titles (about 5%) in the 'other' category, specifically:

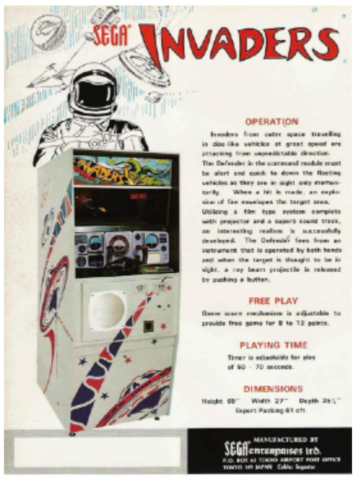
- Lunar Rescue (space themed)
- Love Tester (umm…)
- Jumbo (elephant/circus theme skill game)

… and in 1972, "***Invaders***"

The brochure and cabinet art conveyed the terror of "disc-like vehicles … attacking from unpredictable directions".

The narrative continued:

"The 'Defender' fires from an instrument that is operated by both hands, and when the target is thought to be in sight, a ray beam projectile is released by pushing a button". I'd pay $0.25 for that!



SEGA INVADERS

OPERATION

Invaders from outer space travelling in disc-like vehicles at great speed are attacking from unpredictable direction. The Defender in the command module must be alert and quick to down the floating vehicles as they are in sight only momentarily. When a hit is made, an explosion of fire envelopes the target area. Utilizing a film type system complete with projector and a superb sound track, an interesting realism is successfully developed. The Defender fires from an instrument that is operated by both hands and when the target is thought to be in sight, a ray beam projectile is released by pushing a button.

FREE PLAY

Game score mechanism is adjustable to provide free game for 8 to 12 points.

PLAYING TIME

Timer is adjustable for play of 60 - 70 seconds.

DIMENSIONS

Height 69" Width 27" Depth 35½"
Export Packing 61 cft.

MANUFACTURED BY
SEGA enterprises ltd.
P.O. BOX 63 TOKYO AIRPORT POST OFFICE
TOKYO 149 JAPAN Cable: Segastar

Taito Japan releases "Space Invaders"

6 years later, everything changed. Wikipedia suggests that Space Invaders designer Tomohiro Nishikado drew inspiration from EM target shooting games released in North America in addition to sci-fi narratives including "The War of the Worlds" and the Japanese anime "Space Battleship Yamato" among others.

In the U.S., Midway Games secured a license to distribute Space Invaders, bringing cabinets to every take-out restaurant, bowling alley, and movie theatre in my corner of North America; Taito is said to have sold well over 100,000 units in its first year.

Shigeru Miyamoto, creator of Nintendo Mario, Donkey Kong, and 'The Legend of Zelda' and Eugene Jarvis (Williams Electronics' Defender and Robotron 2084) both cite Space Invaders as an early influence.

Video games slowly displaced Pinball machines, the latter of which occupied at least twice the floor footprint, required routine (if not frequent) maintenance, and had far longer playing times per coin-drop.

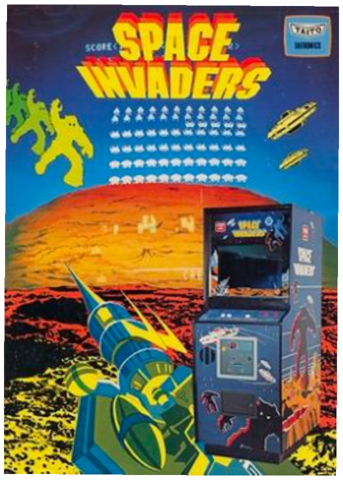
For operators, distributors, and locations, the video game business model was many times more profitable.

Based on the Intel 8080, Space Invaders had a bit mapped 256 x 224 pixel 13" CRT, discreet 555 timer and transistor based sound effects complemented by the TI SN76477 sound IC, and simple 3 button controls.

The atonal 'soundtrack' consisted of four discreet tones which would step downward in frequency, then repeated in coordination with alien movement at a tempo that increased as the aliens advanced.

Periodically, a UFO / flying saucer would appear, complete with sound generated by the Texas Instruments IC. No doubt, this had an effect on the players level of urgency (and blood pressure). Space Invaders was, after all, an all-out assault against humanity.

The 2 MHz. Intel processor was busy and received an assist from a set of hardware shift registers (the 8080 instruction set did not have such instructions). This was instrumental in shifting the now beloved sprites, pixel-by-pixel as the aliens moved horizontally. The cabinet monitor was rotated to portrait and reflected 90' from a mirror, requiring a reverse image on the black and white monitor. Colored cellophane affixed to the mirror provided color filtered light. Space Invaders was a smash hit.



“Back in ’82”

If you played these games when they were new, you can probably rattle off *your* top 10, or at least a quick list that comes to mind. If you didn’t have ‘access’ to the ‘80s or to the games released during the decade, you might find it difficult playing catch up now; there are thousands of titles.

Storied history aside, the list of games produced by year yields a field so far and wide, you’d need to take a year sabbatical to get through 1/2 of it. It’s difficult to tell at this point, but internet sites that catalog such things have numbers upwards of 2,500 in just four or five years of production and for every top title, there were 50 horrible games produced. Operators of this era were purchasing games by the dozen, just to see what took.

I have a fond appreciation for the four Williams games released in 1981 and 1982, specifically, Joust, Robotron: 2084, Defender, and Stargate, in that order.

I’ve also got a thing for Atari’s ‘modern’ games, namely: Marble Madness, Paperboy, Gauntlet, and Championship Sprint.

And who doesn’t have time for Mario Brothers, Donkey Kong, Donkey Kong Junior, and even Popeye?

I almost forgot Missile Command, Centipede, Excitebike, Tempest, Time Pilot, Track and Field, 1942, Kung Fu Master, all of the Pacman variants, and more.

So I’ll draw the line here and discuss 3 from this month’s puzzle and throw in two which were not and we will spend a page per title starting with a favorite.

1. **Crazy Climber** (1980) - an otherwise unknown company named Nichibutsu produced vertical scrolling perfection that was ahead of its time.
2. **Galaga** (1981) - successor of Galaxian; to me, this title is the ultimate stationary shooter; we can debate vs. Gyruss or Satan's Hollow, but considering the year, this game remains unmatched.
3. **Zaxxon** (1982) - Sega came out of nowhere with this diagonal scrolling 3D simulation, expertly done.
4. **Mario Bros.** (1983) - Primitive by design but challenging and the last of the Arcade Mario franchise titles; this two player game could be played cooperatively or… “whoops…”, not !!
5. **Marble Madness** (1984) - Took Zaxxon’s never-been-done 3D perspective to the next level with cartoon whimsey, an amazing soundtrack, and a user interface that demanded physicality and precision.

The remainder of this article will walk through the killer feature or draw that to me, warrants their inclusion on my by-year list of five; I’ll also discuss aspects of the hardware that I found interesting, innovative, or unique, and where applicable, predict the likelihood our seeing a port or ‘inspired by’ on a Foenix platform. Along the way, I’ll provide some web sites and videos that I found interesting as I researched content for this article.

1980 - Crazy Climber by Nichibutsu

It’s hard to believe that between 1979 and 1980, game narratives changed from space and war themed games where the objective was to pilot a ship and “shoot stuff”, to a top selling franchise based on navigating a faceless yellow circle on a dot-eating mission while being chased by Ronald McDonald fry-guy baddies on-a-budget.

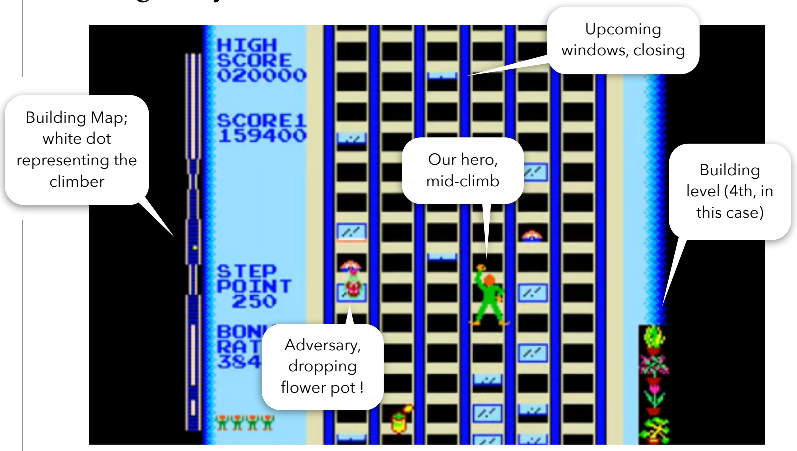
But six months after the Namco release of Pacman, an unheard-of Japanese company released a concept game called “*Crazy Climber*”.

The mission: Climb the building so a helicopter can bring you back to the ground so you can… wait-for-it… climb yet another building.

Along the way, people throw stuff at you; all sorts of stuff; chairs, flower-pots, weight lifting equipment, steel beams, oh… then there is the bird, King Kong, part of an upright piano, and more.

It’s extremely challenging, very colorful, and nicely animated. And like many Japanese games to come, the soundtrack leveraged state of the art audio circuitry with sampled speech calls thrown in. It felt alive.

The controls consisted of two joysticks, one representing each hand, but had no left/right return to center. Instead, you had to switch between frenetic alternating hand climbing, and delicately shifting your hands from ever changing grabbing surfaces, away from closing windows. Residents of the buildings were not neighborly.


Upcoming windows, closing
Building Map; white dot representing the climber
HIGH SCORE 020000
SCORE1 159400
Our hero, mid-climb
Building level (4th, in this case)
STEP POINT 250
Adversary, dropping flower pot !

Crazy Climber - Building #4, annotated

Crazy Climber was a massive work of assembly language (> 13,000 lines) and sprite/tile data. The full source and much of the graphics and other detail (see next page) is expertly mapped out by Rich McFerron on Chris Cantrell's site here.

(You’ll find other goodies on Chris’s site as well, including a few magazine articles that he had published in CoCo / TRS-80 journals in the early ‘80s).

Foenix port feasibility = “High”. I have a high level of confidence that a faithful port is possible, perhaps not the full game (because of the massive amount of work), but a playable game or demo based on the original concept.

Quick look at Crazy Climber Sprite Graphics (two examples)

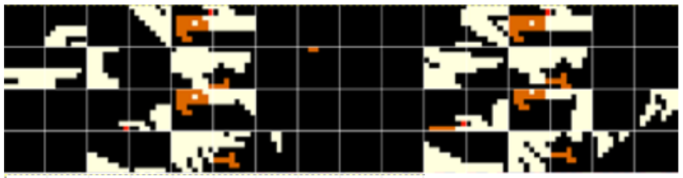
The following represents a few examples of the use of sprites. Let's focus on the Helicopter, and the "Evil Bird".

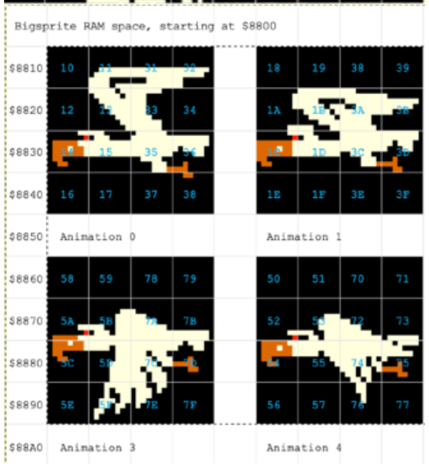
Contrary to end-of-level 'bosses', Crazy Climber greets the player with a hovering helicopter and the opportunity to collect an earned bonus, hear an uplifting song, and celebrate the achievement. Fiddle about for too long and the 'copter will leave with you stranded.

Sixty-four 8 x 8 pixel sprites make up the Helicopter body (assembled below, right) and all possible positions of the main blade and tail rotor which is animated as the 'coper arrives, hovers, and departs (with or without you).

Likewise, the "Evil bird" is also composed of 64 sprites but rather than including animation in a specific region of the "Big" sprite, each frame of animation (4 shown out of the 6 frames) is unique.

"Evil bird":





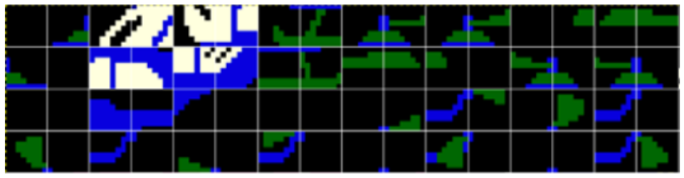
Attention to detail

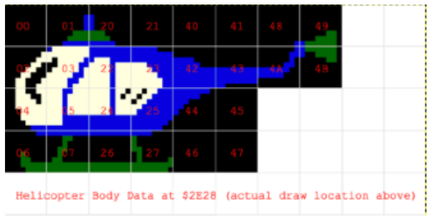
Scanning the hex identifiers (left), you may notice that no two cells are repeated. Despite 'repeating' blank cells above, devs spared no expense in animating to ensure convincing realism.

The following code snippet is associated with bird 'action'. Note the clarity of code formatting and comments. Despite not being 65xx family code, this (for me) is easy to follow. Click here for the source.

```
;========================================================================
; Handle the evil bird
;
; This routine checks to see if the bird is spawned. Once it is, the bird music
; begins to play and the bird is animated and moved.  Finally, a routine is
; called that checks services the bird poop.
;========================================================================
ISR_JUMP3_BIRD:
        LD      A,(GP_82AF)             ; 339A 3AAF82   :.. Get counter
        AND     A                       ; 339D A7       .   Is value zero?
        JR      NZ,Lb642                ; 339E 201D      Y  Nope, go here

        LD      A,(FLOOR_GROUP_IDX)     ; 33A0 3ADC80   :.. Get the current building floor group
        AND     $03                     ; 33A3 E603     ..
        CP      $03                     ; 33A5 FE03     ..  Is the FLOOR_GROUP_IDX != 3?
        JR      NZ,Lb643                ; 33A7 2006      .  Yes, go here

        LD      A,$FF                   ; 33A9 3EFF     >.  FLOOR_GROUP_IDX is 3
        LD      ($82B8),A               ; 33AB 32B882   2.} $82B8 = $FF
        RET                             ; 33AE C9       .

Lb643:  CP      $00                     ; 33AF FE00     .D  Is FLOOR_GROUP_IDX != 0?
        JR      NZ,Lb644                ; 33B1 2006      .  Yes, go here

; We found the bird!
        LD      BC,$406                 ; 33B3 010604   ... Play Bird Music
        CALL    LOAD_SOUND_DATA         ; 33B6 CDBB11   ..A

Lb644:  CALL    INIT_BIRD_GFX           ; 33B9 CDE033   ..2 Initialize the bird graphics
        RET                             ; 33BC C9       .

Lb642:  LD      A,(FLOOR_GROUP_IDX)     ; 33BD 3ADC80   :.. Get the current building floor group
        AND     $03                     ; 33C0 E603     .S
        CP      $03                     ; 33C2 FE03     .S  Is FLOOR_GROUP_IDX= 3
        JR      Z,Lb646                 ; 33C4 2813     (.  Yes, go here

        LD      A,(GP_82AF)             ; 33C6 3AAF82   :.. Get counter value
        CP      $09                     ; 33C9 FE09     ..  Is counter = 9?
        JR      NZ,Lb646                ; 33CB 300C     0\  Nope, go here

        BIT     0,A                     ; 33CD CB47     .G  Is counter even?
        CALL    Z,CHECK_BIRD_POOP       ; 33CF CCEA35   ..q Yes, check bird poop

        CALL    MOVE_BIRD1              ; 33D2 CD4534   .P%
        CALL    BIRD_ANIMATION          ; 33D5 CD0435   .Tq Animate the bird
        RET                             ; 33D8 C9       .

Lb646:  CALL    BIRD_ANIMATION          ; 33D9 CD0435   .Tq Animate the bird
        CALL    MOVE_BIRD2              ; 33DC CDBA34   ..%
        RET                             ; 33DF C9       .   End ISR_JUMP3_BIRD
```

"Helicopter":





Quick listen to Crazy Climber Audio

The General Instruments AY-3-8910 is driven by (at least) 1,400 lines of assembly language located from $4000 - $4FFF (4K assembled). This memory space contains init and register drivers but also, data streams embedded in ".db" statements within the source.

In addition to sound effects and a few musical jingles, there are a number of voice calls urging the player to "go for it!", exclaiming "yeah!", and screaming "ouch!" upon being struck by falling objects. And of course, the "oh nooooooo…!!!" when the player is knocked to the ground. Click here for samples.

Amusing (?) excerpt from pg. 4 of the service guide

1) ANTAGONIST
A bad man who drops things from the windows.
2) MANY DROPPING THINGS
Dangerous objects that Antagonist drops.
3) THE CONDOR
She drops eggs and excrements.

Final thoughts and a note on gameplay

Considering it was 1980, the quality of animation and audio was quite high. But this game has also been characterized as having a "steep learning curve".

One of the internet-based fan sites characterizes a first time user as likely to be seen:

"… shaking the joysticks every which way and unable to figure out how to achieve even the most basic movements." This is part of the appeal. Crazy Climber was a great game, and not well known.

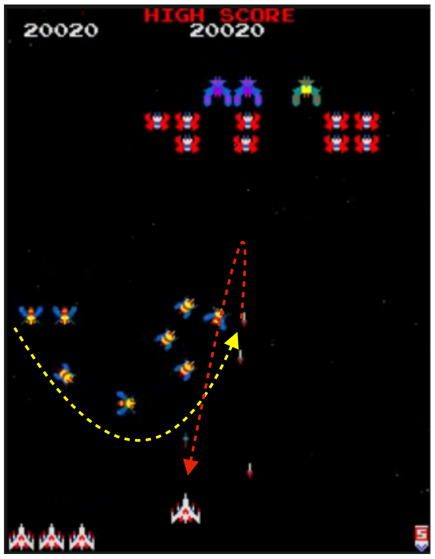
1981 - Galaga by Namco

Sixteen months following the release of Pacman, Namco would produce a game that was possibly their 2nd most successful title, "*Galaga*". With deference to DigDug and Pole Position (both Namco titles licensed by Atari), Galaga was a top earner and one of the few games to approach the success of Pacman, though a distant 2nd.

Galaga succeeded Galaxian, but the leap in game play and refinement between the two titles felt like 3-4 years of evolution, not just two.

The unfair comparison leaves Galaxian feeling like a sophomoric effort; quirky and uninspiring, but there was no denying, it was a first.

With Taito's Space Invaders lead, many games were produced using the "stationary shooter" *formula*: a single, player controlled ship at the bottom of the screen moving along the x-axis (more or less); aliens descending against a backdrop of stars, dropping or shooting projectiles and swarming in order to create havoc and mayhem.



Galaga stage 5 - (bees flight path annotated in yellow and red)

Phoenix, Moon Cresta, and Gorf (the latter of which, strangely, had Space Invaders and Galaxian within) all used this formula.

Galaxian was the first game with full color sprites but to some extent, was developed in haste;

What *had* changed at the turn of the decade was advances in computer technology and development practices.

Galaxian and Galaga used same Zilog (Z80) CPU and both ran at 3.1 MHz. But Galaga had *three* of them (and thus could support 64 'player/enemy' sprites). One CPU ran the main game logic and coordinated master control over the other two; a second Z80 managed graphics and enemy movement, and finally, a third, managed audio. Galaxian only had 8 sprites and group of small missile objects driven by the sole Z80 CPU.

The Galaxian player might not have realized it, but the 8 sprite limitation required developers to leverage a combination of stationary bit shifting 'tiles' (as Space Invaders did) for non-moving enemies, and one of the 8 sprites for an enemy once in motion.

Galaxian is a difficult and frustrating game but loved by many (not me). On the other hand, Galaga was a gem in every regard. The animation, object movement, music and gameplay were all perfectly coordinated and Namco repurposed the 'intermission' concept from Pacman, extending it to a playable skill / challenge stage where the player could earn points without risk of losing a life.

They also added a tractor-beam capture/docking feature that allowed the player to double-up (with some risk) and borrowed the progressive scoring feature from Galaxian, awarding extra points for defeating all members of a sortie attacking in formation.

(Here is a fascinating interview with Shigeru Yokoyama, Namco designer; as interesting as it is revealing.)

Namco, wary of piracy and clones, included a number of custom chips across generations of their boards, but they didn't invent the technique (look for an article in an upcoming issue for more on copy protection). What Namco did do is create custom ICs that to this day, are still challenging experienced engineers.

YouTuber "ajcrm125" documented his efforts to clone a Pole Position IC using *Verilog* (the same software that Stefany has leverages for FPGA development) to model and then mimic the Namco chip. Worth a watch!

Sprite movement

We mentioned Galaga's sprite engine in prior discussion. From a multiprocessing perspective, there is something special about what Namco pulled off (again, relative to early days of arcade). Using 3 x Z80s was surely a help, but that aside, the ability to run not only 64 sprites, but stationary animation flawlessly, was the game's crowning achievement. Think about the trajectory of the bees and how they run tight but slightly different paths looping down towards the player in coordination (see yellow dashed line to the left) and then join the ranks above. Sometimes, however, one or two would peel off and adopt an angry dive-bombing trajectory (annotated in red). Namco took no shortcuts.

Game Play

What one thing made Galaga great? Everything! Namco did what everybody was doing, attempting to capture the attention of players, baiting them in with a given formula, but then hooking them for life, with graphics, audio, precision, and enough variety to build a following. It was the Pacman formula all over again.

Audio Soundtrack

Galaga employed the same audio hardware shared by DigDug and Pacman whose core was a 4-bit Waveform Generator running at the system clock of 3.1 MHz; primary tones were square waves but had a smooth tonal quality to them. Other chip features were used for sound effects.

Three voices yielded such lovely, if not regal overtures. Learn to play the theme song for yourself here or if you are a super-fan, preview then buy the music on Apple Music here.

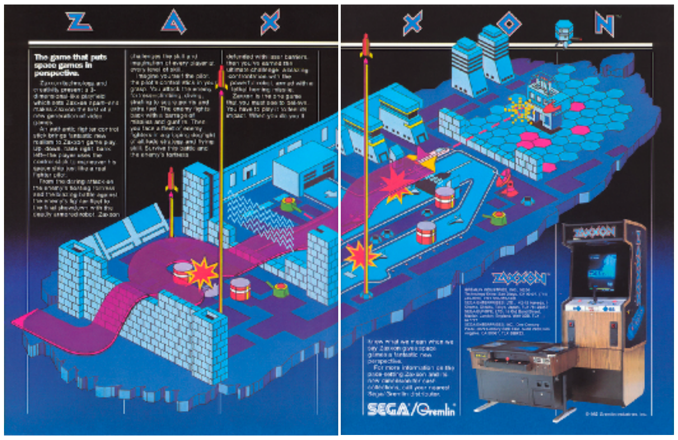
Foenix port feasibility = "Moderate". Can one 14 MHz. CPU match 3x Z80s, running at a fraction of the clock speed? Stefany says 'yes'.

1982 - Zaxxon by Sega

Zaxxon was one of the most popular arcade machines in 1982, but only for *one* month. Depending on continent, Zaxxon's short lived 'day in the sun' was outshined on both ends of its single month reign by one of the most beloved games of all time; Namco's Ms. Pacman. It was hardly fair considering the head start that the original Pacman had.

But Zaxxon rewrote the book on the space shooter genre adding a 3rd dimension and a visually reorientated perspective to a crowded field of video games that was just starting to dabble in perspective and shadows. Sprites were not even operating in multiple fields of depth or layers in any useful way through 1981.

The wow factor of 3D and multi-dimensional scrolling was amplified by Zaxxon's icy blue color scheme and bitmap detail that, at the time, felt like perfection in every regard. (however, the fact that it shipped in a woodgrain cabinet /see pic/ was just plain weird)



Zaxxon was a challenging game to play as well, especially at *that* moment in time. Experienced players were used to fixed-shooter x-axis-only movement, or two-axis limited motion, but they were not ready for this.

Each level began with an attack on the enemy base, continued through an open air space, and concluded with a more difficult fortress and an end-of-level boss.

Another *first* for a video game was the famous primetime TV commercial. Sega (owned by Paramount Pictures at the time) reportedly paid $150,000 for a highly-produced 30-second TV commercial. Ports to every platform imaginable were popping up including the monochrome TRS-80, the TRS-80 CoCo, Atari 2600, ColecoVision, Coleco Adam, and Japanese platforms, just to name a few, and there was even official Zaxxon merchandise like this Milton Bradley board game.

Sega reused the technology and graphics engine on Super Zaxxon and for the game *Congo-Bongo* a year later, but this level of visual gameplay perspective would otherwise not be improved upon in such a meaningful way until Marble Madness was released by Atari in 1985.

From a technology perspective, Sega leveraged the same Z80 CPU running at 3 MHz. as others but they certainly made the most of it. It also had a massive pile of components to create at least a dozen sound effects from discreet electronics, all CPU switched through an 8255 peripheral interface IC. See the left side of the schematic on pg. 134 here for this detail.

A quick word on Graphics capabilities

For as beautiful as Zaxxon was graphically, it only supported a 256 x 224 screen resolution, had a 9 bit color palette (512), but impressive capabilities otherwise:

- Tile-map planes: 2 layers (foreground, background), 8×8 tiles, 4 or 8 colors per tile, tile flipping, vertical/horizontal/diagonal scrolling, isometric perspective
- Sprites: 4 or 8 colors per sprite, flipping & shadows
- Sprite sizes: 8 and 32 heights, widths of 8, 16 and 32 pixels
- Line buffer: 256 sprite pixels/texels per scanline, 8 (32-width) to 32 (8-width) sprites per scanline

All in all, the Zaxxon technology was an engineering marvel for Sega, but with the exception of two or three titles mentioned, it was bespoke to the extent that it was not considered a "platform".

Sega "System 1" would later be released and was leveraged across more than a dozen Sega games. Not to be confused with Atari's 1985 vintage "System 1" which was used for Marble Madness, Road Runner (meep-beep), RoadBlasters. Oh, and not to be confused with Gottlieb's "System 1" solid-state pinball board system, used across a dozen machines (I own two of these; a story for another time).

Game Play - a pre flight-simulator simulator

There is something about flying a plane and needing to pull back on the yoke with all of your might to gain enough altitude to NOT crash into 'that' mountain, building, or in this case, an electrified force field wall.

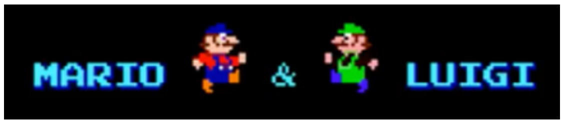
Of course, I've never flown an airplane, let alone, a rocket powered space jet. But if Zaxxon is any indication of difficulty, well… it's difficult. Banking, being wary of your altitude without ground perspective, avoiding projectiles launched from underground, and being shot at from the side adds to the difficulty. Between bases, the free flight portion of each game level challenged the player with numerous enemy craft flying at varied elevations. As much as it was a relief to escape the fortress of death, free flight was also harrowing.

At the time of Zaxxon's release, a small company named SubLogic had already been pushing an 8-bit micro Flight Simulator for the Apple II (later sold by Microsoft). The first incarnation of FS featured rudimentary line drawn scenery from a first person point of view. The tech was nowhere near being able to depict a smooth animated view of aircraft from a relative vantage point five hundred feet away, but Zaxxon did. Of course, it was all fun and games, but an impressive technical feat.

1983 - Mario Bros. by Nintendo

Shigeru Miyamoto is widely credited for being the creative force behind the arcade version of *Donkey Kong* but he subsequently developed the Mario franchise, bringing *Super Mario Bros.* into homes via Famicon (Japan) and the Nintendo Entertainment System (NES) Console in the rest of the world.

Prior to the NES introduction (1985), Nintendo released the arcade game *Mario Bros.* It followed *Donkey Kong Junior* and was the 3rd game in the series; but 'Bros.' was not a smash hit. What it did do, was to put Mario back in the drivers seat, and introduced a new character, his brother Luigi, and cooperative play.



Gameplay and Nintendo Gaming in the '80s

Nintendo was doing something unique during this period. After an absolutely dreadful start at arcade in the late '70s, they had started to standardize the feel and level of graphics and audio quality that carried across the small quantity of games produced. They also resisted the temptation to use anything but a 4-direction joystick with a single button, and had used the same cabinet across all games through *Punchout*.

The Mario character attributes of the original came to Mario Bros. with familiar running and jumping skills; but they were expanded upon graphically to make the animation and game play more interesting. Making contact with a flipped over turtle or lobster would trigger a kicking motion; running in one direction then opting to stop or change direction would cause your player to skid briefly (or slide uncontrollably if the surface was frozen); most significantly, jumping was transformed into a power move whereby contact with another object would flip it over, nudge it, or used to collect points.

This new ability could be also be used to trigger a new on-screen feature, the "POW". These capabilities, though still controlled via a single jump button, would further evolve and become key in the release of *Super Mario Bros.* on the NES and in other titles. It was a new dimension of fun derived from the same old controls.

Oddly, the next series game, *Donkey Kong 3*, abandoned Mario and Luigi. Instead, Nintendo introduced an exterminator named "Stanley the Bugman". Huh?

Nintendo must have seen the beginning of the end. Following the release of Donkey Kong 3, they would only produce Punch Out, Super Punch Out and an absolutely dreadful arcade game called *Arm Wrestling*.

They had quietly been focusing the majority of their efforts on developing and then repackaging the Famicon system into the NES for U.S. and European markets, an investment that would yield unimaginable value and transform Nintendo into an entertainment powerhouse.

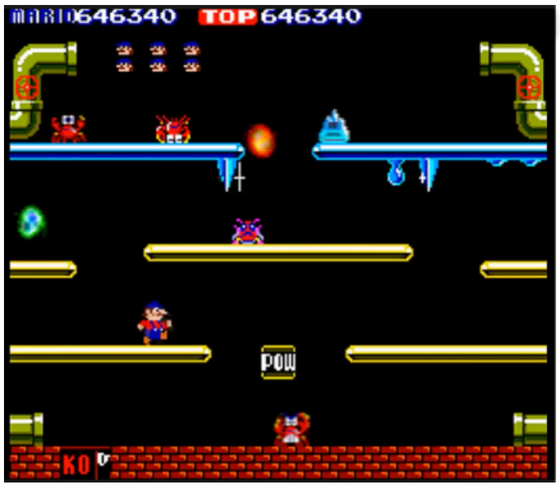
Technology, what technology?

Similar to other arcade companies of the era, Nintendo was iterating their CPU board-set from game-to-game and as late as 1983, still had not established a 'System'. This would change with the advent of the Nintendo 'Uni' and 'VS.' platforms, and ultimately the 'PlayChoice 10'.

However, over four years, Nintendo used the same Z80 based board, the "Donkey Kong Board", for four games (one per year). The first, a little known stationary shooter called *Radar Scope* looks interesting in YouTube videos, but didn't appear to get much love.

Donkey Kong, Donkey Kong Jr., and Donkey Kong 3 were the other three titles that ran on this board. (counts of machines produced vary wildly.)

Nintendo had a different board for the 1982 *Popeye* game (one of their few licensed titles). *Sky Skipper* (1981), released in Japan, also used this board. See this YouTube video. Sky Skipper was recently added to the Nintendo Switch Store. It's a rather rough game but you can see the early rudiments which would find their way into several games including the tile scrolling that would eventually become a platform staple on the NES system.



Mario Bros. screen resolution of 256 x 224 pixels was identical to the Donkey Kong and Junior games that were released prior, but the color palette appears wider. Again, there is little written about this hardware.

Nintendo's success was never about unique and fancy cabinets or flashy controls. They thrived on gameplay, characters, and later, storyline; the technology was never that important.

What is certain is the fact that Atari, Namco, Williams (including the Bally Pinball division), and scores of other arcade design and manufacturing companies are long since gone, and Nintendo had enough focus, discipline and vision to know when to get out and to somehow, find a way to parley a handful of beloved characters into an overwhelmingly successful console (the NES), titles for it, and then a series of handhelds, the SuperNES, the GameCube, Wii, Switch, and the merch that goes with it.

Foenix port feasibility = "High". Easy lift, given the time and perseverance to get it done.

1984 - Marble Madness by Atari

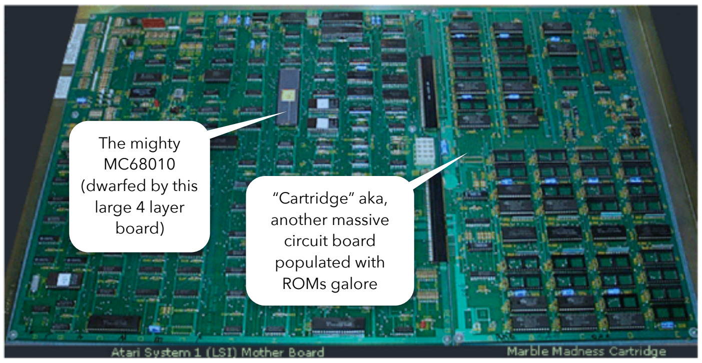
Marble Madness is similar to Crazy Climber in many ways. It has a control scheme that looks familiar enough but is more difficult than it looks; it was ahead of its time relative to competitors, and you either loved it or you walked right past it.

But Marble Madness is different than Crazy Climber because it was produced at the end (of the end) of the arcade market crash. You just didn't see games like this anymore; they cost too much to produce and did not sell.

There is something else about this title that is not very well known; it was developed on the iconic Digital Equipment Corporation (DEC) PDP11/780 mini-computer (*the* 1 MIPS reference platform), it was the first at Atari to have been developed in the 'C' language, and did so on a relatively new microprocessor (the Motorola 68010). It also was the first title released on the Atari "System 1" platform, the first to use true stereo, and to do so using the Yamaha YM2151 (OPM).

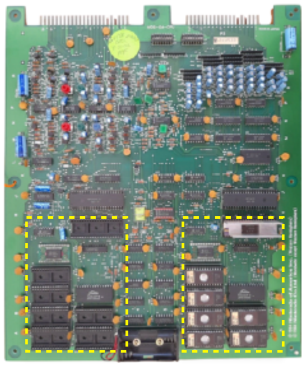
About System '*x*'

Atari developed the System 1 concept to allow cabinet conversions from title to title. The thought was to have a common main board and 'cartridges'. But as it turned out, the cartridge was nothing more than yet another gigantic board containing dozens of ROMs.


The mighty MC68010 (dwarfed by this large 4 layer board)
"Cartridge" aka, another massive circuit board populated with ROMs galore
Atari System 1 (LSI) Mother Board
Marble Madness Cartridge

With graphics, audio, and gameplay improving, the need for more computing power was evident and this was Atari's answer.

Meanwhile, across the Pacific, Nintendo was working on a similar approach but at small scale; with ROMs first, then **actual** cartridges (PlayChoice 10), but with much simpler games and lower tech, Nintendo's vision would pay dividends for years to come. On the right is a pic of what Nintendo was producing in 1984-1985. Nintendo *VS.* hosted two sets of ROMs (2 games) at a fraction of the cost.


"Nintendo VS". - approx. to scale w/ the Atari System 1 board above

Super Mario Bros. (right) unpopulated ROM sockets for a 2nd game (left)

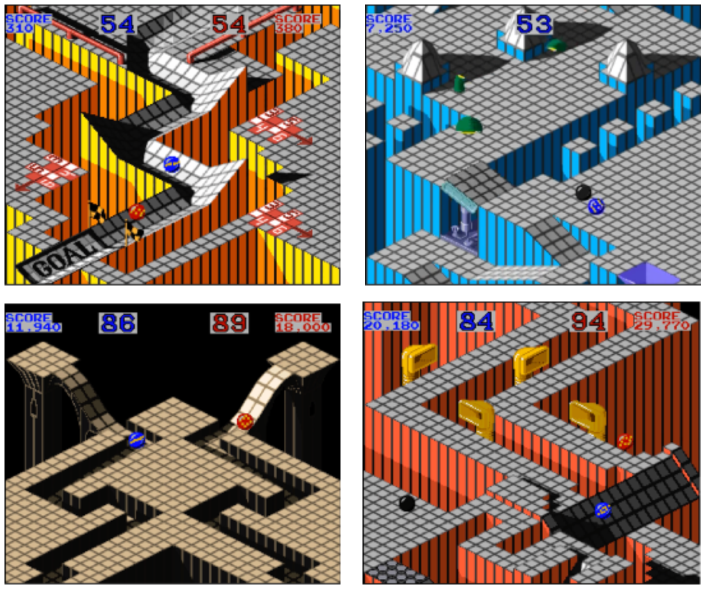
Of course, following System 1, Atari moved back to a monolithic board design, leveraging the CPU, graphics, and audio technology of System 1. This is what Ed Logg's famous *Gauntlet* ran on. It was still expensive.

In parallel, System 2 was in the works and Atari would go on to produce a concept which was similar to System 1, except based on the unheard of DEC T11 CPU. (ancestor of the DEC Alpha chip) System 2 ran Paper Boy, 720', Super Sprint a few other titles.

Oh, the game

It was amazing. Otherworldly. Mark Cerny is said to have been influenced by MC Escher and was fascinated with 3D technology. (see link* below for a great video)

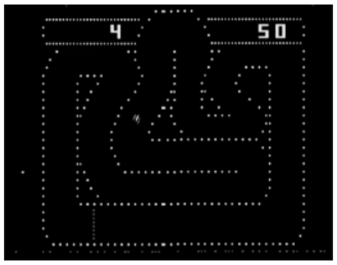
The most engaging aspect of the game (to me) was the way the music and overall experience was woven into the visuals. Little did we know (at the time) that the resolution was 'only' 336 x 240 but the color palette was large (256 colors defined out of total available of 1024).



The end

In mid-1984, Warner sold the home computing and electronics division of Atari to Jack Tramiel for $50 (yes, fifty dollars) and $240 Million of promissory notes and stock, giving Warner a 32% stake in the company. Warner also sold the Arcade division to Namco in 1985.

And just like that, Atari (this incarnation of Atari) ended where it started. With a driving game. Of course, the name 'Atari' would live on and Arcade eked onwards; but it was never the same.


Gran Trak 10
The beginning ('74')

Super Sprint via Namco / Midway
The end ('85')

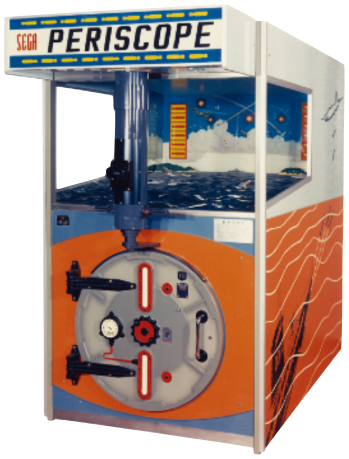
* More from Mark Cerny, recorded at GDC 2011

Arcade before Arcade

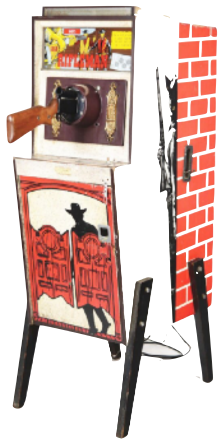
A visual tour of a portion of Sega's Electro-mechanical lineup from the '60s and '70s

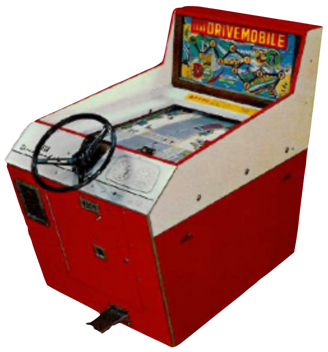
(Sourced from SegaRetro.org, an outstanding resource; and specifically this page)

Also, SegaDriven.com put together a nice tribute to many of these games in this video

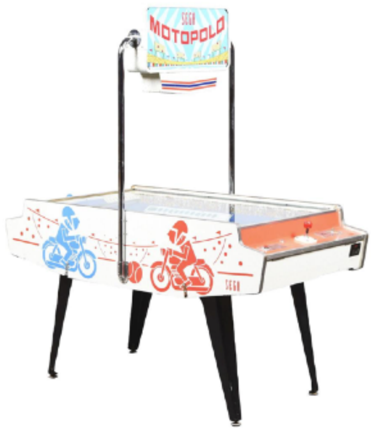


Periscope ('66)
(SEGA's first title)

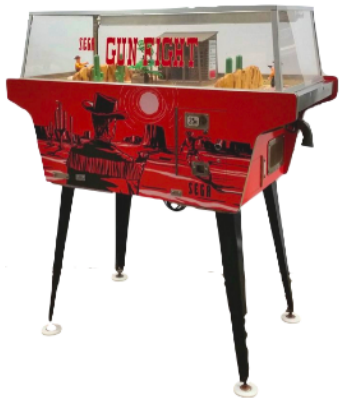

Rifleman ('67)



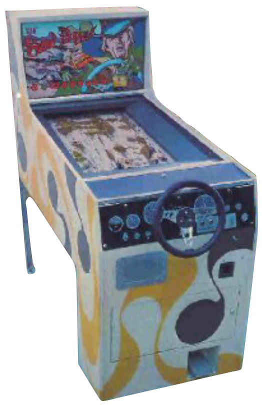
Drive Mobile ('68)

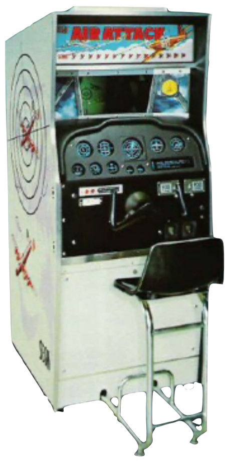


Moto Polo ('68)

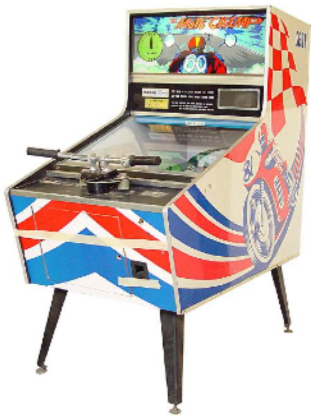


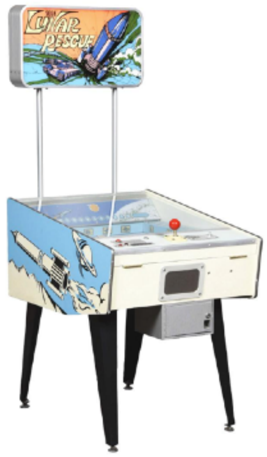
Gun Fight ('69)


Sand Buggy ('71)



Sega Air Attack ('72)


Moto Champ ('73)



Lunar Rescue ('73)

Pacman

A quick look at one aspect of the original design plus a Foenix Ms. Pacman preview

Q&A with the developer of a working demo for C256 platforms

How did they do that?

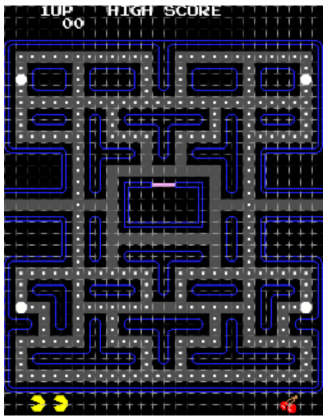
How did Namco pull off such a masterful arcade experience given the computer technology available in 1979 into 1980?

Pacman was an early Zilog Z80 CPU title, but the Z80 processor was by no means new. Released in 1975, it had been used on everything from late '70s CP/M machines, the 1977 TRS-80 Model 1, dedicated video terminals from Zenith/Heathkit, and even early gaming consoles such as the Bally Astrocade (1978).

Pacman's clock ran at 3.072 MHz., which was considered fast when compared to the 1.934 MHz. clock of the prior generation video games (such as *Carnival* by Gremlin / Sega). And while that may sound fast (versus the 1 MHz. 6502 many of us are accustomed to), it's not. The Z80 had many advantages over the MOS CPU but common opcodes took 2x or 3x more clock cycles to complete the same task; so the speed diff was negligible.

What Pacman did have*, was help from proprietary hardware based sprites (the platform supported 8 at 16 x 16 pixels) and 3 channel audio; but what Pacman really had was clever developers that quickly surmised that the 224 x 288 bit mapped screen was 'too' hi-res to be managed on a pixel-by-pixel basis. And without the utility of collision interrupts (which would be common later), game logic had to correlate telemetry between Pacman and the Monsters efficiently in order to preserve gameplay and leave cycles for other important work. It also had to do this reliably, and you're about to see (or perhaps already know) that this was a challenge.

The method leveraged divided the navigable portion of the playfield into a 'coarse' grid of locations which are 8 pixels apart (*x* and *y*). Pacman and his adversaries traverse the maze smoothly (pixel by pixel) but collisions are checked at 1/60th of a second intervals, and only by comparing the location of Pacman and any ghosts in proximity against the 298 odd tiles where a collision may *possibly* occur (far less work than having to calculate a field across 64,512 pixels).



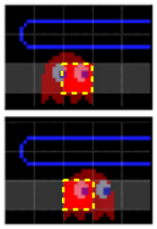
Movable objects may only exist on one of the grey 'navigable' tiles (298 of them)
Image from the Pacman Dossier

Oddities

Because of this scheme, *two* of three gameplay collision anomalies have been experienced by many of us, and the *third* (the "pass-through") is legendary, but rarely seen.

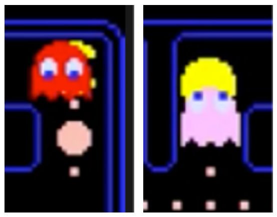
Caught, not caught - As explained in the *Pacman Dossier*, Pacman gains speed when the player actuates a turn prior to arriving at an intersection.

Ghosts on the other hand, make right angle turns anytime they turn corners. Pacman also changes his speed (as do monsters) depending on level, whether eating a dot or not. This give and take presents opportunities to escape monsters grasp (usually Blinky), and at times, you appear to be caught, but somehow just manage to pull away. This occurs because the two objects (Pacman and Blinky) do not *yet* occupy the same coarse tile location even though they are clearly overlapping by a few pixels on the screen.



Blinky *occupies* the same location while moving smoothly across ~2 full tiles

The second (**variable catch**) is also related to the way that Monsters and Pacman occupy positions. It occurs when you've actually been caught by a monster (or have collided with one), but the amount of contact varies. The graphic below shows two examples of this (each screenshot was taken immediately after capture). On the left, Blinky, moving from left-to-right has just about crossed paths with Pacman and in fact has already focused his eyes downward (was about to make the turn). In the 2nd example, Pinky caught Pacman as he was about to transition to a turn to the right; if he had 1 more pixel lead, he might have survived!



Finally, there is the **pass-through;** which occurs when Pacman and a monster are approaching each other and happen to cross coarse boundaries outside of the 1/60 of a second frame interrupt where collisions are checked.

Patterns and Strategy

I first played Pacman in late 1980/early 1981; it was on a cabaret machine in the snack bar of a Modell's department store in New York. I recall watching older kids run specific patterns that they either made up, learned from each other, or appropriated from books.

At the time, there wasn't much thought invested in studying the behavior of the monsters or developing a strategy; for me it was all about trying to remember what worked the last time, or just winging it. But many years later, there have been online guides published that map algorithms of the original code. They are fascinating to read whether you are a developer or just love arcade games. Here are two:

This link will bring you to an interesting guide of patterns; it also discusses variations, and other tips.

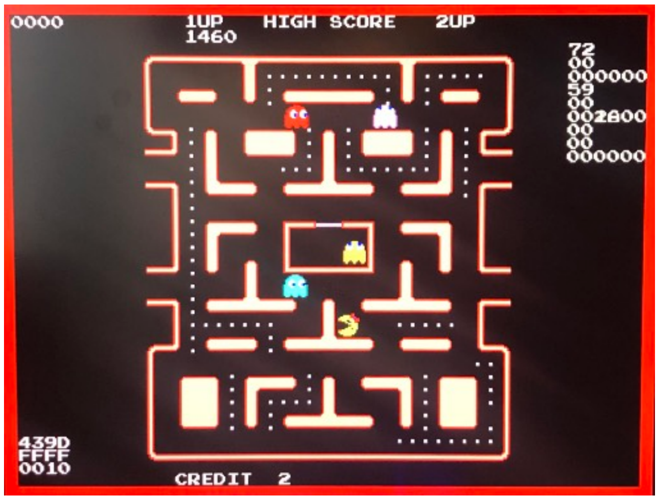
* see this link for Alessandro Scotti's excellent hardware capability overview

And the end-all/be-all is the Pacman Dossier, which was invaluable in writing some of this article. I won't go into detail on how the monster behavior is managed, but urge you to have a look. It was written and/or compiled by Jamey Pittman (a Twin Galaxies registered champion) and is outstanding work. Jamey also plays guitar!

(A disassembled Pacman source can be browsed here.)

Ms. Pacman, coming to a Foenix C256 near you

Those following the Foenix Discord channel may have seen a work in progress port of Ms. Pacman for 65816 platforms. It is currently a playable demo, graphically accurate in every regard (as far as we can tell).



Based on a publicly available Z80 source listing, the developer is working on the Foenix port as a personal challenge. We had a chance for a quick chat and I am pleased to share the dialogue.

As part of the October update, we've also posted a .hex file of one of the later versions of the demo to the Foenix Marketplace. Keep in mind as you play this, it's merely a demo and just for testing (plenty of bugs present, some of which add to the fun).

Q: Is your code written in Assembly, C, or something else and which compiler/assembler did you use?

A: It's written in [65816] assembly using Merlin32 by Brutal Deluxe.

(Editor's note: Apple II aficionados will remember the original ***Merlin*** *Assembler, which was developed by* Glen Bredon*, a UC Berkeley math professor who later taught at Rutgers in New Jersey. When Glen passed away, his wife graciously turned his work over to the public domain)*

Q: You mentioned you took the Z80 source, was that just for the AI or monster movement / behaviors or was the Z80 assembly more or less [complete and] directly transferrable to rewriting it in 65816. Do you think that the classic patterns work?

A: It's directly ported from original code… basically everything. If you were to put a 65816 inside the Pacman cabinet, the game would run.

Q: Did you have to do anything special to negotiate interrupts or the diff in speed between the 3 MHz. and the Foenix or do timers take care of that for you?

A: I'm trying to get it as close as possible to the original. Ms Pacman arcade had patches to try and reduce the usage of the patterns that were used in the original Pacman. The only thing that is challenging [to model precisely] is the random number generation; the Z80 has a built in random function, and Ms. Pacman has as second random function that samples the 64k memory space in the hardware. Our 64k space doesn't match the original, so the random numbers will vary.

The C256 is clocked much faster than the original hardware so there are no issues there. The Pacman / Ms. Pacman main loop runs on the *vblank* interrupt on the original hardware, so I'm just doing the same thing on the Foenix. The game will be a hair slower since the original hardware I think is something weird like 60.6 Hz. and the Foenix is 60 Hz.

[from an audio perspective] The DAC in the original Ms. Pacman hardware can do 3 voices at 96 KHz. And the DAC on the Foenix is a single voice at 48 KHz., so there will be a challenge software mixing the 3 channels.

Q: I was about to ask you about audio. Did the original use PCM based sounds.

A: I haven't done the audio yet, so we'll see. It has a very small PCM wave table. I'm looking forward to hearing those sounds.

Also the original hardware uses 8 x 8 tiles, and 16 x 16 sprites. So on the Foenix I'm in 800 x 600 mode, my tiles at 16 x 16 (I'm just doubling up the pixels), and the same with the sprites (32 x 32), doubled up from 16 x 16; so the code just treats it as if it's 8 x 8 tiles, and 16 x 16 sprites.

Basically at the higher resolution, the Foenix native sizes can be made to look half sized.

Q: Great info, thanks. Final question, what is next for you after you complete this; another classic game or will you do something net new? Also what Foenix hardware do you own or have on order?

A: I have an FMX, and a U+. I'm unsure what is next. The source code to Merlin 16 is floating around, it would be interesting to bring it over to the Foenix, so that Ms. Pacman could be assembled on the actual hardware instead of cross-assembled. But I'm not sure what's next. One thing at a time. Probably the original Pacman would go very quickly. But we'll see. I have a Gen-X on order.

* Interested in the origins of Ms. Pacman? See this talk by Steve Golson of GCC, recorded at Game Developers Convention 2016

Just 4 Fun - This month's puzzle

A 'tear-out' activity book exercise for the kid in all of us.
'MAD Magazine' meets 'Children's Highlights Magazine' meets Foenix

Foenix hardware evolution

Long time Foenix watchers have seen pre-FMX hardware development morph into the FMX (the first GA product), then to the C256 U+ and so on.

Some models have quite a bit in common with others, and some, not so much; but they all share a common DNA and many of the same components.

Your mission: Examine page 17 closely and take action based on the instructions below. (download the hi-res version from the Foenix Marketplace for best results).

This is not just fun and games. This month, **Foenix Rising is sponsoring a contest with prizes:**

- Puzzle #1: Visually inspect the A2560K circuit board picture and **identify** at least 10 aspects of the board picture that seem out out of place, peculiar, or that just do not make sense considering what you know about the specifications of the platform; the **highest correctly identified tally submitted by December 15th will win a custom made mini-ITX case and a picoPSU power supply** similar to the case pictured on pgs. 1, 30, and 32; with it, you'll be able to get your F256 Jr. powered up and into action quickly!

  You'll need your own keyboard and monitor, but much of the header wiring will be complete and the included power supply is plug-and-play. Item will be shipped world-wide at no cost in the month of December or January, pending receipt of my own prod version, required in order to guarantee compatibility and fit. To win, you must have submitted an order for a F256 Jr.
- Puzzle #2: Solve, **by modding** the A2560K "circuit board" to transform it into something that it is not. **Correctly completed puzzles receive a 40 page, spine bound compilation of Foenix Rising *Beginner's Corner* and related articles** from issues #1 through #3, printed on high quality paper stock. It's the perfect quick start guide to get moving if you are new to Foenix.

To submit an entry:

Email me at the address noted on pg. 2 above or reach out via Discord. In the immortal words of Gene Kranz, "good luck to all of you".

About:

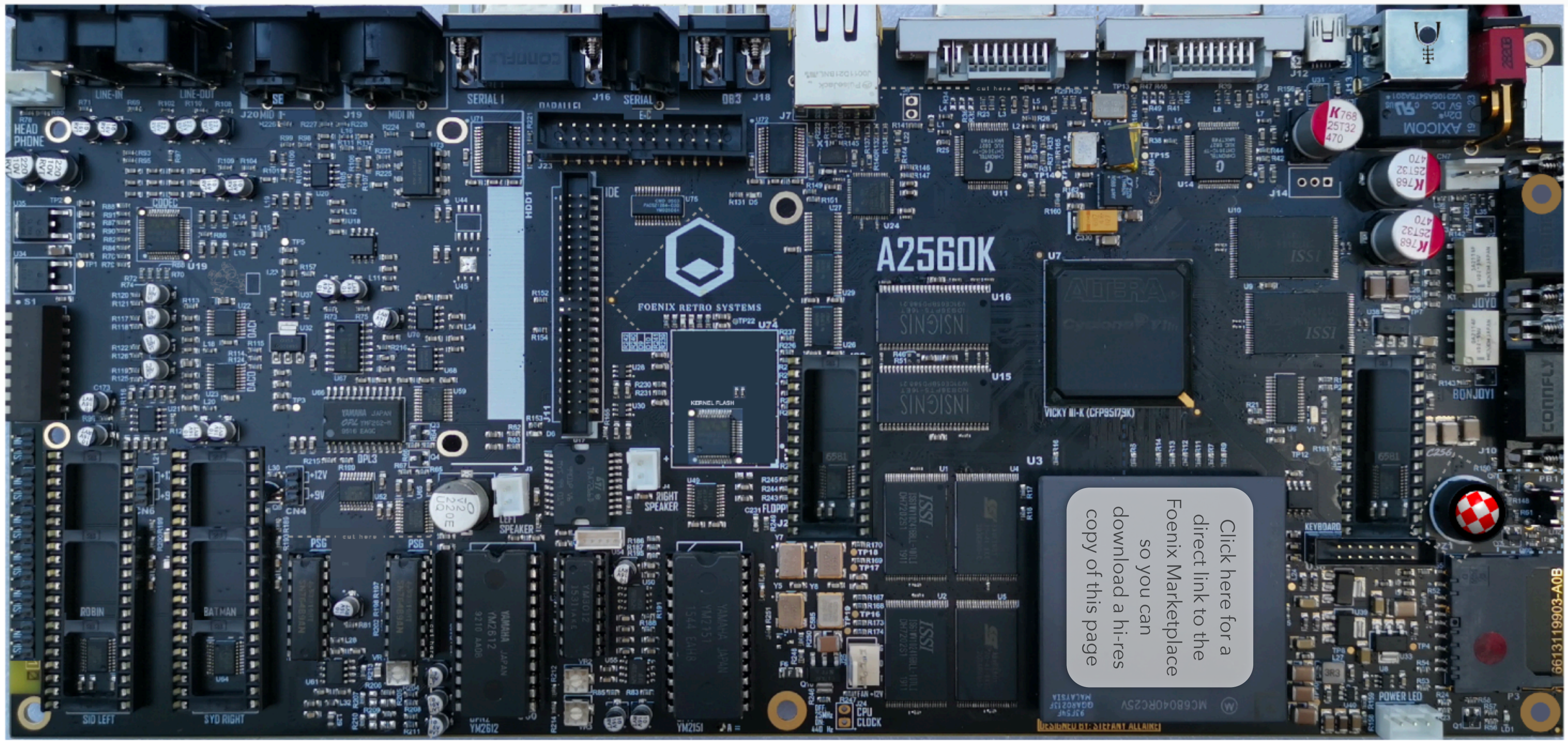
- The original Mad Magazine was published in the 2nd half of the last century and had an unusual off-brand of humor, often including biting political and social commentary. As a youth, I recall one of my Uncles having a stack of these. Other than Alfred E. Neuman, mostly, I remember one very specific feature, printed on the inside of the back pages. All I can say is *you'll know it when you see it*.
- Children's Highlights was a family run business for many years and continues to publish *actual* puzzles and activities including the type featured in Foenix Rising such as word search, crossword, and cut-out activities.

Puzzle Hints (may apply to either puzzle):

- Board 'modifications' may be necessary (BYO scissors or a graphics editing tool); Foenix designs often come with unpopulated sockets and options; consider how you might modify the circuit board in order to transform it into something that it is not.
- Consider how one or more of the cut-out components located at the bottom of the circuit board picture might be leveraged in a modified design.
- Circuit board designers often embed hidden messages on board labels, can you find any that look out of place? (I will share that despite Stefany Allaire's appreciation of humor and irony, all of the out-of-place notations are my doing and part of the puzzle).
- Foenix systems include standard ports for as much connectivity as possible, but not all boards support all peripherals. What does this board appear to feature that may not belong? What can be done to change this?

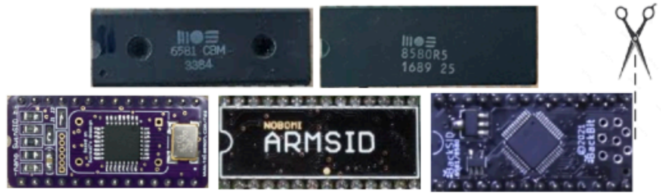
Solve this months puzzle by finding at least 8 thinz wrong with this picture (not including the typo) scissors are required...

Just 4 fun from Foenix Rising

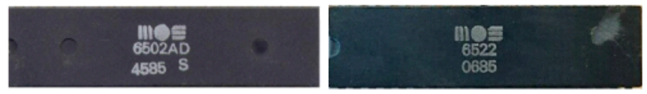


Click here for a direct link to the Foenix Marketplace so you can download a hi-res copy of this page

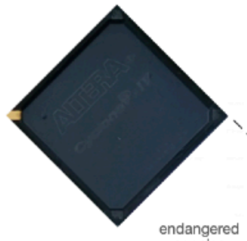
The machine she designed for her own use, a transformation that few saw coming. Perfect fit and built for function; a development workstation that also says "I want to play"



The A2560K will work with most any SID (just set voltage on CN6 & CN4 or CN7 & CN5 on a JR :)
(one SYD nobody could work with was Pink Floyd's original frontman, Syd Barrett, whose antics were dramatized in this movie: T h e _ a _ _)



This *dynamic-duo* joined forces on 8-bit machines in the 70s and 80s
(not to be confused with a the crime fighting duo from this 60s Superhero TV Comedy): _ a _ _ a _ .


endangered species

Date and Time on your Foenix

Written by Ernesto Contreras

On pg. 31. of our August issue, we reviewed an issue of COMPUTE magazine from 1980 and touched upon an obscure '80s product, the *Hayes Stack Chronograph*. As part of the discussion, we made a passing reference to the Foenix RTC circuit. This month, contributing developer Ernesto Contreras is back with a closer look at how this Texas Instruments IC is implemented on Foenix Computers. In this article, we will learn about BCD (binary coded decimal) and interface details of the TI chip. Ernesto has included a BASIC816 example that allows you to set and read the time/date yourself. Look for this program (as "I03P20A.BAS") on the Foenix Marketplace, and begin to experiment for yourself.

Introduction

Do you know the internal date on your Foenix computer? I'm willing to bet, you don't; after all, it's not very straightforward how to query your FMX or Foenix U / U+ computer about what date they think it is.

But let's take a few steps behind, first you need to know that your Foenix computer, unlike other 8-bit retro computers of the time (C64, Apple, Atari), is equipped with a Real Time Clock (RTC) chip, the Texas Instruments BQ4802.

https://www.ti.com/lit/ds/symlink/bq4802ly.pdf

This simple chip has a real time clock that keeps track of the date/time, even when you power off your computer (assuming you installed the required battery). The chip also allows you to program alarms on specific dates/times that can trigger an interrupt on the Foenix, but that's outside the scope of this article.

So, now you know that your Foenix has this chip, and if you have played a bit with the integrated BASIC816 you might notice that you can query the time on your computer using the command **GETTIME$**, that responds by printing the time onscreen in the format `HH:MM:SS`.

```
PRINT GETTIME$(0)

11:05:23
```

I know this is hardly impressive, but this opens a few more questions:

- *How do I get the Date?*

  Answer: BASIC816 has no command to show us the Date, but it can be queried from the chip with a few **PEEK** commands and a bit of number decoding magic.

  (nothing is straightforward in this life…)

- *How do I change the time/date if it's wrong?*

  Answer: As you might guess it by now… BASIC816 doesn't have a function to do this, but you can **POKE** values to update the time and date *(if you know what values to poke!)*

Well before going into peeking and poking our way around, you might notice my senseless humor in the comments in parenthesis, well those comments are there because the chip keeps track of dates and time using BCD encoding, and if you don't know BCD encoding, you're in luck! since we are providing a brief primer on what BCD is and how to use it.

BCD Encoding

BCD stands for "*Binary encoded decimal*" and it is a way of representing numbers in memory. In this encoding each digit is represented by a fixed number of bits. In this case the chip uses a *packed BCD* format (or simply packed decimal). In this format each of the two nibbles (4 bits unit) in each byte represent a decimal digit.

Decimal digit	BCD encoded value bits 0-3			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Packed BCD has been in use since at least the 1960s and is implemented in all IBM mainframe and other* CPUs since then.

The maximum BCD encoded number that can reside in a byte is 99 (since each nibble holds a digit from 0-9).

The implementation in the Texas instruments chip is big endian, with the more significant digit in the upper half of each byte and the less significant on the lower half.

Nibble 1	Nibble 0
0010	0001

* fun retro fact - The Nintendo NES used the Ricoh 2A03 processor which is a modified version of the 6502 core with additional functions added. Ricoh *disabled* the BCD functionality; some say to accommodate other capabilities, others say to avoid copyright infringement!

Example: the number 21 as represented in Packed BCD:

```
Value encoded in Byte  : 33
Binary representation  : 0010 0001
Decoded Decimal        : 2    1
```

Shifting and masking operations are used to pack or unpack a *Packed BCD* digit. Other bitwise operations are used to convert a number to its equivalent bit pattern or reverse the process.

Knowing all this we can provide an example in BASIC on how to encode/decode BCD using:

- Division and multiplication for shifting
- Bitwise AND operations

Here is how to encode a number between 0-99 in *Packed BCD* using BASIC816:

```
v%=value: REM value to convert (0-99)
BCD%=int(v%/10)*16+(v%-int(v%/10)*10)
REM encoded byte in BCD%
```

Note: Multiplying by 16 (or 2 ^ 4) is the equivalent of shifting a number 4 bits to the right (moving it to the upper nibble)

Example: encode the decimal number 21

```
V%=21
BCD% = int (21/10)*16 + (21 - int (21/10) * 10)
BCD% = (2*16) + (21 - 2*10)
BCD% = (32) + (21-20)
BCD% = 32 + 1 = 33
```

Here is how to decode a number in *Packed BCD* using BASIC816:

```
BCD%=BCDvalue: REM encoded BCD number read from
a Packed BCD byte
v%=int(BCD%/16)*10+(BCD% and 15): REM unencoded
value in v%
```

Note: A bitwise AND of the current byte against 15 preserves the

	Base 10	*Binary Bits*							
		Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Current byte	*167*	*1*	*0*	*1*	*0*	*0*	*1*	*1*	*1*
Bitwise Operation Mask	*15*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
Result	*7*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*

value of bits 0,1,2,3 and turns off bits 4,5,6,7

Example: decode the *Packed BCD* number 33

```
BCD%=33
V%= int(33/16)*10+(33 and 15)
V%= int(2.0625)*10 + (1)
V%= 2*10 + 1 = 21
```

Now that we know how to interpret BCD numbers let's go ahead and see where the Real Time Clock data resides.

RTC Registers

For the Foenix U / U+,   the following Memory address hold the registers of the Real Time Clock:

Register	**Address**	**Description**
RTC_SEC	$AF0800	Seconds Register
RTC_SEC_ALARM	$AF0801	Seconds Alarm Register
RTC_MIN	$AF0802	Minutes Register
RTC_MIN_ALARM	$AF0803	Minutes Alarm Register
RTC_HRS	$AF0804	Hours Register
RTC_HRS_ALARM	$AF0805	Hours Alarm Register
RTC_DAY	$AF0806	Day Register
RTC_DAY_ALARM	$AF0807	Day Alarm Register
RTC_DOW	$AF0808	Day of Week Register
RTC_MONTH	$AF0809	Month Register
RTC_YEAR	$AF080A	Year Register
RTC_RATES	$AF080B	Rates Register
RTC_ENABLE	$AF080C	Enables Register
RTC_FLAGS	$AF080D	Flags Register
RTC_CTRL	$AF080E	Control Register
RTC_CENTURY	$AF080F	Century Register

Addresses $AF0800 - $AF080A and $AF080F use values in BCD encoding

Date and Time Components

The registers listed above can be used to put together the date and time, to be explicit on how it needs to be done here's a brief example.

Date / Time:

```
2022/09/10  5:02:00
```

Date Component	Century	Year	Month	Day
Example	20	22	09	10
Address	$AF080F	$AF080A	$AF0809	$AF0806

Time Component	Hour	Minute	Second
Example	5	02	00
Address	$AF0804	$AF0802	$AF0800

Remember, the value from these registers is encoded as BCD

To summarize

I. To read the date and time you need to:
   a. PEEKing the appropriate RTC registers
   b. Decode the BCD value in these registers to Decimal
   c. Show selected Date/Time Components

II. To change the date or time you need to:
   a. Encode the Components you are going to update (year/month/day, etc…) in BCD
   b. POKE the BCD values into the appropriate RTC registers

Sample Program

Finally, here is a sample BASIC program that reads & updates the *Date / Time* using the RTC in your Foenix.

If you choose to update the Date/Time, the new values should survive after powering off your computer (assuming a battery is present). You should be able to read the new defined values after you power down and up again with this program.

```
5 CLS:REM read date/time from RTC chip
10 c%=PEEK(&haf080f):y%=PEEK(&haf080a):REM Century / Year
20 m%=PEEK(&haf0809):d%=PEEK(&haf0806):REM Month / Day
30 h%=PEEK(&haf0804):n%=PEEK(&haf0802):REM Hour / miNute
40 s%=PEEK(&haf0800):f%=PEEK(&haf080d):REM Second
50 yr%=(INT(c%/16)*10+(c% AND 15))*100:REM decode BCD numbers
60 yr%=yr%+(INT(y%/16)*10+(y% AND 15)
70 mo%=(INT(m%/16)*10+(m% AND 15))
80 dy%=(INT(d%/16)*10+(d% AND 15))
90 ho%=(INT(h%/16)*10+(h% AND 15))
100 mi%=(INT(n%/16)*10+(n% AND 15))
110 se%=(INT(s%/16)*10+(s% AND 15))
120 IF (f% AND 1)=1 THEN 140
130 PRINT "Battery Failing, date/time might be invalid!":GOTO 120
140 PRINT "Battery Status Ok!”
150 PRINT "time stamp:";yr%;"/";mo%;"/";dy%;"     ";
160 PRINT ho%;”:";mi%;":";se%
170 PRINT "press y to change date/time”
180 GET k$: IF k$<>"y" THEN 340
190 INPUT "Enter Year”;y%
200 INPUT "Enter Month”;m%
210 INPUT "Enter Day”;d%
220 INPUT "Enter Hour”;h%
230 INPUT "Enter Minute”;n%
240 c%=INT(y%/1000)*16+(INT(y%/100)-INT(y%/1000)*10):REM BCD Century
250 y%=y%-INT(y%/1000)*1000:REM get decimal Year
260 yr%=INT(y%/10)*16+(y%-INT(y%/10)*10):REM BCD year
270 mo%=INT(m%/10)*16+(m%-INT(m%/10)*10):REM BCD month
280 dy%=INT(d%/10)*16+(d%-INT(d%/10)*10):REM BCD day
290 ho%=INT(h%/10)*16+(h%-INT(h%/10)*10):REM BCD hour
300 mi%=INT(n%/10)*16+(n%-INT(n%/10)*10):REM BCD minute
310 POKE &haf080f,c%:POKE &haf080a,yr%:POKE &haf0809,mo%
320 POKE &haf0806,dy%:POKE &haf0804,ho%:POKE &haf0802,mi%
330 GOTO 10340 END
```

Available on the Foenix Marketplace as “I03P20a.BAS” (click here)

Ernesto's Foenix Sprite Editor v1.1

A quick look at the new features and a closer look at palettes

Last month, we examined the v1.0 release of the Foenix Sprite Editor for C256 platforms. The short recap, in case you missed it: Foenix Sprite Editor is an easy to use tool, purpose built to support the capabilities of Foenix platforms, and a natural evolution from the original sprite editors used on vintage platforms. I continue to use it and found it handy for animating sprites such as those featured in Beginner's Corner.

In mid-August, Ernesto reached out with information on a 1.1 release and we are here to talk about it.

As prior, the distribution is packaged within a PC / Mac compatible `.zip` file and the v1.1 distribution looks to have at least 3 files updated.

- `SPREDIT.BAS` - includes bug fixes and support for new functions including changes to the default palette
- A new icons file (now named `ICONS2.SPR` to accommodate the new colors)
- An updated manual

New features

v1.1 includes several new features (most associated with color tools) as follows:

Color pickup - there is now a color *pickup* function. Similar to MacOS, Windows, or open (GIMP, for example) graphic applications, this feature is useful when needing to select a color you've already used on the edit grid. Right clicking actuates this function, selecting the values as used in the identified cell.

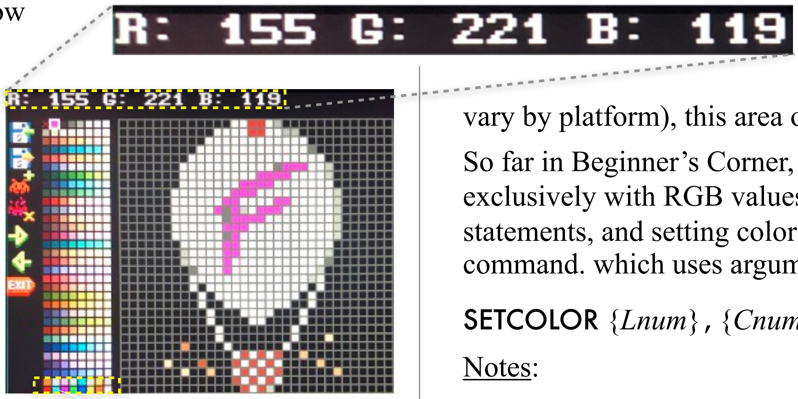
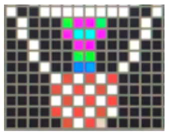

R: 155 G: 221 B: 119

figure 21a

RGB value display - choosing precise custom colors is now easier, thanks to a new visual indicator of RGB values noted on the status bar at the top of the screen; see the highlighted section called out in yellow in figure 21a above.

New default palette - there is an updated default palette which includes changes to dozens of colors, a new 'basic' 16 color palette (located at values $F0 - $FF the last two row of the color picker), and smoother gradians across a range of base colors.


Sprite editor view w/ v1.1 palette

... should look closer to this (as displayed)

Note: If you've built sprites using the v1.0 palette and neglected to save the palette, you'll notice that your sprites look a bit odd; see the pic of the Foenix Balloon basket to the right as an example.

To revert back to the original colors, you've got three choices.

a) check page 23 below for values you can plug into the **DATA** statements within `SPREDIT.BAS`. Once modified, you should re-save the program with a different name.

b) head to the Foenix Marketplace and look for the file `SPREDTV1.PAL`, then use the load palette function each time you use the Sprite Editor.

c) use the new the RGB custom color edit sliders to dial-in a precise colors that meet your approval where needed.

Always use the save palette features (see next page).

Reintroducing Palettes

It is important to differentiate the way in which the Foenix Sprite Editor instantiates its palette versus the way that BASIC816 manages color data, versus the manner in which you will need to address Foenix (Vicky II) registers, if poking values from BASIC or storing values from assembly language.

Regardless of the method leveraged, once stored, it will be difficult to impossible to read stored values because palettes are stored in VRAM memory space and in most versions of FPGA code (which will vary by platform), this area of memory is write-only.

So far in Beginner's Corner, we've been working exclusively with RGB values encoded in BASIC **DATA** statements, and setting colors using the **SETCOLOR** command. which uses arguments as follows:

SETCOLOR {*Lnum*}, {*Cnum*}, {*Rval*}, {*Gval*}, {*Bval*}

Notes:

Param	Description	Valid values
Lnum	LUT Number	`0 .. 7`
Cnum	Color Number	`1 .. 255`
Rval	Red value	`0 .. 255`
Gval	Green value	`0 .. 255`
Bval	Blue value	`0 .. 255`

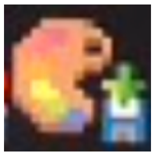
Vicky II supports 8 color look up tables (#0 through #7) for sprites, tiles, and bit mapped graphics as well as two color LUTs for text modes (#8 for 15 foreground colors and #9 for 15 background colors).

Actual RGB color will vary slightly based on your monitor type and settings but will generally map to the 24-bit Pantone style colors found on numerous sites. Browse the web and you'll see what I mean.

Foenix Sprite Editor `.PAL` files

When the editor initiates, it loads the icons file into memory along with a set of machine code functions, but as of the v1.1 release, color palette data is still populated from **DATA** statements in the BASIC portion of the program (lines 2000 - 20500).

When you use the save-palette-to-disk function (the icon to the right), the editor creates a 1K file (256 colors x 4 bytes per color).

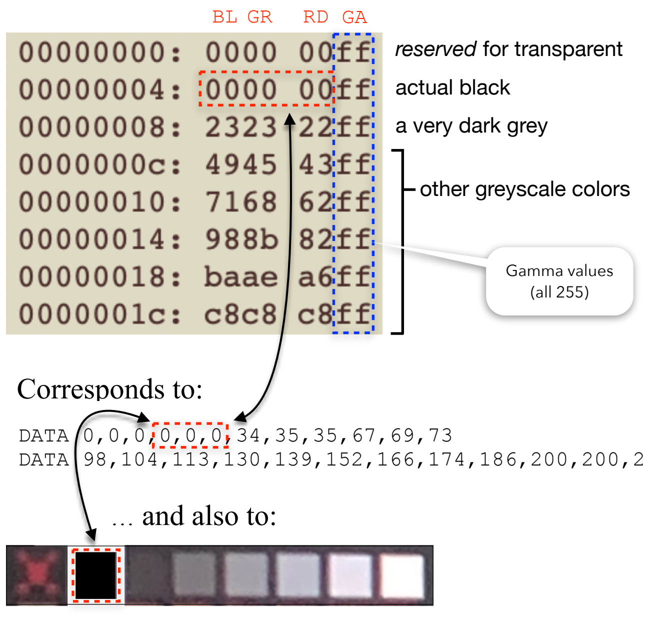

save palette icon

The `.PAL` file is a binary data file with a repeating pattern of values representing the GAMMA, RED, GREEN, and BLUE of colors in the palette but [important point], **stored in reverse** in order to match Vicky II's memory map and therefore, be load-ready. See below.

The BASIC **SETCOLOR** command, on the other hand, masks this peculiarity from the user; (colors are noted in RGB order and there is no mention of GAMMA values; they are dealt with behind the scenes).

Worth noting that the GAMMA value is not currently implemented in Vicky II and is not used by **SETCOLOR**, but it still consumes 256 bytes of the 1024 byte `.PAL` file, it exists in memory, and is represented in the table below in the rightmost column.

The following diagram explains this encoding scheme. Here, we tie the memory map (and file format) to BASIC statements, to the first 8 colors of the color picker in the editor. Again, note that GAMMA, red, green, blue are reversed (see headings).



```
00000000: 0000 00ff
00000004: 0000 00ff
00000008: 2323 22ff
0000000c: 4945 43ff
00000010: 7168 62ff
00000014: 988b 82ff
00000018: baae a6ff
0000001c: c8c8 c8ff
```

Corresponds to:

```
DATA 0,0,0,0,0,0,34,35,35,67,69,73
DATA 98,104,113,130,139,152,166,174,186,200,200,200
```

… and also to:

Using a `.PAL` in your own program

Two steps are all that are required in order to instantiate a custom palette from this load-ready file.

1. Load into SRAM memory using a **BLOAD**, such as:

```
BLOAD "MYCUSTOM.PAL", &h100000
```

2. Move from SRAM into VRAM with **MEMCOPY** in the same way that you would use it for moving sprite data after load (however, using the **LINEAR** directive because this is numeric data, not a shape*):

$$\text{MEMCOPY LINEAR \&h100000, 1024 TO LINEAR } \underbrace{\text{\&hAF2000}}_{\text{address}}\text{, 1024}$$

As discussed previously, once established (or in this case, 'loaded') we need to associate the object with a LUT via **SPRITE**. The following table details LUT and addresses; we used $AF:2000 in the example above.

LUT	Address range
0	`$AF:2000 - $AF:23FF`
1	`$AF:2400 - $AF:27FF`
2	`$AF:2800 - $AF:2BFF`
3	`$AF:2C00 - $AF:2FFF`
4	`$AF:3000 - $AF:33FF`
5	`$AF:3400 - $AF:37FF`
6	`$AF:3800 - $AF:3BFF`
7	`$AF:3C00 - $AF:3FFF`

Manipulating `.PAL` data from BASIC

If you wanted to do it the old fashioned way, you can use the eight line BASIC program below to load a `.PAL` file from disk and then transfer it to VRAM memory by iterating from 1 to 255. (remember, color 0 is transparent and will be conveniently skipped based on the logic and range leveraged below)

Doing it the hard way is a learning opportunity. Here we use **PEEK** statements wrapped in a **FOR** : **NEXT** loop:

```
10 BLOAD "MYCUSTOM.PAL", &h100000
20 FOR x% =1 to 255
30 base%= x%*4+&h100000
40 b%= peek(base%)
50 g%= peek(base%+1)
60 r%= peek(base%+2)
70 SETCOLOR 0,  x%, r%, g%, b%
80 NEXT
```

Of course, you can make this 'your own' and improve it by inserting **PRINT** statements with an onscreen counter, a bar graph or something else; how about 64 balloons that transform from grey to full and varied color while ascending or descending based on color temperature like Galileo's thermometer?

What about performance?

Relative to the MMU function (used by **MEMCOPY**), BASIC code of this type is slow but in most cases, we will only be doing it once at the start of our program.

If interested in more? I'll meet you in *Beginner's Corner*, just two pages down.

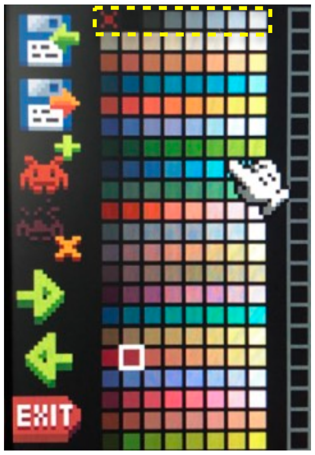
* the diff being that *shapes* are two dimensional (*x*-width by *y*-height) and therefore have 'stride'; see pg. 18 of issue #2

Foenix Sprite Editor - Color Table (from v1.0)

This table documents the full color palette LUT from the initial release of the Foenix Sprite Editor and provides a means to retrieve the data behind selected colors

	Color 0			Color 1			Color 2			Color 3		
Color #	Red	Green	Blue	Red	Green	Blue	Red	Green	Blue	Red	Green	Blue
0 - 3	0	0	0	0	0	0	34	35	35	67	69	73
4 - 7	98	104	113	130	139	152	166	174	186	200	200	200
8 - 11	98	93	84	133	117	101	158	140	121	174	161	137
12 - 15	187	175	164	204	195	177	234	219	201	255	243	214
16 - 19	88	49	38	115	61	59	136	80	65	154	98	76
20 - 23	173	110	81	213	141	107	251	170	132	255	206	127
24 - 27	0	39	53	0	56	80	0	77	94	11	102	127
28 - 31	0	111	137	50	140	167	36	174	214	136	214	255
32 - 35	102	43	41	148	54	58	182	77	70	205	94	70
36 - 39	227	120	64	249	155	78	255	188	78	255	233	73
40 - 43	40	43	74	58	69	104	97	95	132	122	119	153
44 - 47	134	144	178	150	178	217	199	214	255	198	236	255
48 - 51	0	34	25	0	50	33	23	74	27	34	89	24
52 - 55	47	105	12	81	136	34	125	164	45	166	204	52
56 - 59	24	31	47	35	50	77	37	70	107	54	107	138
60 - 63	49	142	184	65	178	227	82	210	255	116	245	253
64 - 67	26	51	44	47	63	56	56	81	64	50	92	64
68 - 71	65	116	85	73	137	96	85	182	125	145	218	161
72 - 75	94	7	17	130	33	29	182	60	53	228	92	95
76 - 79	255	118	118	255	155	168	255	187	199	255	219	255
80 - 83	45	49	54	72	71	77	91	92	105	115	115	127
84 - 87	132	135	149	171	174	190	186	199	219	235	240	246
88 - 91	59	48	60	90	60	69	138	82	88	174	107	96
92 - 95	199	130	108	216	159	117	236	197	129	255	250	171
96 - 99	49	34	42	74	53	60	94	70	70	114	90	81
100 - 103	126	108	84	158	138	110	192	165	136	221	191	154
104 - 107	46	16	38	73	40	61	102	54	89	151	84	117
108 - 111	185	109	145	193	120	170	219	153	191	248	198	218
112 - 115	0	46	73	0	64	81	0	81	98	0	107	109
116 - 119	0	130	121	0	160	135	0	191	163	0	222	218
120 - 123	69	49	37	97	74	60	126	97	68	153	121	81
124 - 127	178	144	98	204	169	110	232	203	130	251	234	163
128 - 131	95	9	38	110	36	52	144	70	71	167	96	87
132 - 135	189	125	100	206	151	112	237	182	124	237	212	147
136 - 139	50	53	88	74	82	128	100	101	157	120	119	193
140 - 143	142	140	226	156	155	239	184	174	255	220	212	255
144 - 147	67	23	41	113	43	59	159	59	82	217	74	105
148 - 151	248	93	128	255	125	175	255	166	197	255	205	255
152 - 155	73	37	28	99	52	50	124	75	71	152	89	90
156 - 159	172	111	110	193	126	122	210	141	122	229	154	124
160 - 163	32	41	0	47	79	8	73	93	0	97	115	8
164 - 167	124	131	30	150	154	38	180	170	51	208	204	50
168 - 171	98	42	0	117	59	9	133	79	18	158	101	32
172 - 175	186	136	46	209	170	57	232	210	75	255	246	79
176 - 179	38	35	61	59	56	85	86	80	111	117	104	110
180 - 183	145	122	123	179	151	131	207	175	142	254	223	177
184 - 187	29	44	67	46	61	71	57	77	60	76	95	51
188 - 191	88	113	44	107	132	45	120	158	36	127	189	57
192 - 195	55	36	35	83	57	58	120	76	73	148	93	79
196 - 199	169	109	88	191	126	99	215	147	116	244	163	128
200 - 203	45	75	71	71	101	90	91	123	105	113	149	125
204 - 207	135	174	142	138	193	150	169	209	193	224	250	235
208 - 211	0	27	64	3	49	95	7	72	124	16	93	162
212 - 215	20	118	192	64	151	234	85	177	241	109	204	255
216 - 219	85	71	105	118	93	115	151	116	136	185	140	147
220 - 223	213	163	154	235	189	157	255	213	155	253	247	134
224 - 227	29	29	33	60	49	81	88	74	127	121	100	186
228 - 231	149	133	241	169	150	236	186	171	247	209	189	254
232 - 235	38	36	80	40	51	93	45	61	114	61	80	131
236 - 239	81	101	174	82	116	197	108	130	196	131	147	195
240 - 243	73	33	41	94	65	74	119	83	91	145	96	106
244 - 247	173	121	132	181	139	148	212	174	170	255	226	207
248 - 251	114	28	3	156	51	39	191	90	62	233	134	39
252 - 255	255	177	8	255	207	5	255	240	43	255	255	255
	Color 252			Color 253			Color 254			Color 255		

Two rows in the table to the left represents 8 colors or ***one*** row below (the yellow dashed line)



Picture of (part of) the v1.0 Sprite Editor color palette. NOTE: this was taken with an iPhone so is not true color.

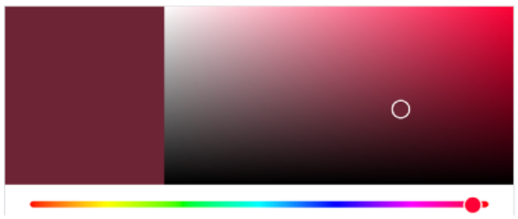
Each grey shaded row represents the *start* of a row in the palette above.

The white square cursor above is on the 2nd position of the 17th row.

To work your way from the color picker to RGB values, count rows (*r*) and subtract 1, multiply by 8, then add the left-to-right position (*p*) minus 1: ((*r* - 1) * 8 + (*p* - 1)) to locate the color # that fits within the range in the leftmost column. Or in this example: (17 - 1) = 16 * 8 = 128 + (2 - 1) = 129

This yields: **R = 110; G = 36; B = 52**

And just to double-check, we can Google "RGB 110 36 52" and find the expected result:



Yes! That looks about right to me…

Motion and commotion in 25 lines or less

In this month's column, we close the multipart BASIC816 "Foenix Balloon" series by trading memory to avoid complexity; We will also discuss BASIC language coding 'style' and animation. This will be the last of the BASIC816 series for now, but we'll return two issues from now with some Assembly Language fun.

In issue #1 of this publication, we introduced sprites and discussed how to define and move them from point 'a' to point 'b'. In the article, passing reference was made to "non-linear" movement and "animation" as advanced topics and we promised to revisit them.

In issue #2, we wrote and used a few simple BASIC816 programs to analyze sprite LUT data and then wrote some code to affect simple color-based animation; we also wrote code to move the Foenix Balloon diagonally around the screen, obeying the edges of the screen in an infinite flight path.

(we also failed to implement one feature, to animate a simulated flame).

In this issue, we will redeem ourselves by doing the following:

- leverage different sprite images for 'powered' flight versus unpowered (descending/floating) flight
- animate a flame, using several pre-defined sprites and carefully selected colors from a specific color LUT.
- further animate the Balloon when 'landing', using a sequence of images that when swapped frame-to-frame, exhibit the appearance of a hard landing
- leverage some amount of randomness in path of descent to not only break from the previous formula of linear diagonals, but also, to simulate a quicker descent and increased randomized horizontal motion at higher (numbered) '*y*' pixels (which are lower on screen)
- add additional variability by dynamically modifying part of the color palette with a minimum of code
- most importantly, do all of this in 25 lines of BASIC code (not counting the palette or sprite data load/setup)

What are we covering in this installment

Before we get to the code and methods leveraged, we have two matters to cover.

1. Discuss the updated sprite set - in the introduction, we mentioned the memory vs. complexity trade-off; we've 'improved' the base image and expanded the single sprite to a 12-sprite set.
2. Discuss programming 'style' and the fact that we broke a few of the cardinal rules before we've even established them.

And then we will discuss the algorithms and approach leveraged for this month's program. Finally we will talk about where we are heading from here.

Sprite Animation

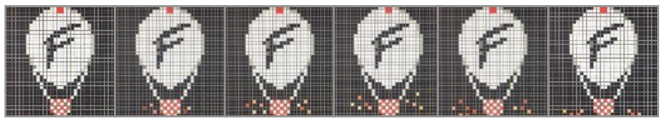
In issue #2 we animated the beacon at the top of the balloon but did so using color LUT manipulation instead of flipping between multiple sprite images.

This month, we will do a bit of both.

To begin, the original sprite was modified to use a solid color across the Foenix 'F' (versus the choice of transparent/color $00), and we fixed a few mis-clicks in the selection of the original and v2 sprites (see figure 25a of the August issue and related text).

Next, we used the Foenix Sprite Editor copy/paste function to duplicate the base image and then chose a series of white, yellows, oranges, and reds that would be convincing when animated to simulate a flame. The end-goal (as prior) was to animated the flame only when ascending. We used a 6 sprite set for this.

With a 2nd set of 6 sprites, we chose one frame of the basic balloon without a flame for use when descending (also used when stationary / landed) and then copied and modified it five times to create a 'splashdown' effect, simulating dust and dirt kicking up upon landing.



See figure 24a for a closeup of the first three frames of the basket.

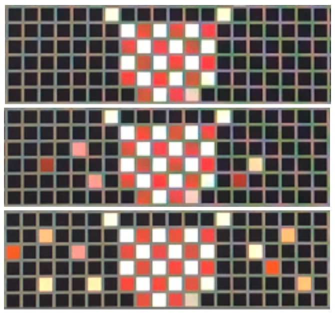


fig. 24a

It's an extremely simple effect (with only five frames used), but will be convincing once put into action.

We could have used 512 sprites **just** for this animation instead of five. Remember, Vicky II supports 64 *simultaneously displayed* sprites; the number of sprite *definitions* is limited only by the amount of memory we have (or are willing to dedicate to it). As you will see in the code, re-pointing a visible sprite to a different image is as simple as plugging a different address in the **SPRITE** command as follows:

$$\text{SPRITE 0, 0, } \underbrace{\text{\&hB00000+(spr\%*1024)}}_{\text{address}}$$

A word on memory, complexity, and the trade-off

In total, the new sprite-set consumes 12 x 1024 bytes of storage/memory or 12K in total. (the file saved by the Foenix Sprite Editor is actually *this* amount + 1 byte; the additional byte contains the value 12, which represents the number of sprites within).

The 12,288 bytes of data is more than 1/3rd of the entire usable memory inside of a Commodore 64, but then again, the Commodore did not support sprites of such density and high color. As we discussed in issue #1, Commodore sprites were only 63 bytes in size (versus 1024 bytes on Foenix). There is something to be said about necessity, and an embarrassment of riches.

Computing has always been a trade-off between the triad of CPU power, memory (and/or storage) capacity, and input/output bandwidth / speed, and the bottleneck moves as technology improves and economics shift. If the bottleneck did not appear *somewhere*, we would accomplish an infinite amount of work, instantly. Computing is also a trade-off between complexity of code and resources consumed. It's nice to have choices.

As retro platforms go, Foenix platforms put us in great shape; even on the middle-of-the road Foenix C256 U+, we have 8x the # of sprites, from 8 to 32 times the amount of memory, greater than 8 times the CPU speed and power not to mention an acceleration engine (for lack of a better word) to coordinate direct memory access, graphics, and the rest (in Vicky II).

The F256 Jr. (see pgs. 29-32) rightsizes the display back to 1982 resolution but adds full color depth, powerful graphics, and other features of the bigger Foenix platforms. It is well appointed.

Code style and conventions

The BASIC programming language never imposed discipline in the way that procedural languages ultimately did. BASIC also did not require strict type casting; programs "just worked".

BASIC was invented at Dartmouth University in 1964 and truly was intended to be "for the masses". Have a quick look at this 50-year celebratory (4 minute) video.

Fun fact, the Foenix Marketplace BASIC icon (figure 25a) was derived from Dartmouth's green celebratory graphics from their golden anniversary campaign in 2014. Dartmouth also produced a more comprehensive recount of the anniversary in the "Birth of BASIC", here.

fig. 25a

Despite extensions (and in many cases, abuses) by every 8 bit computer manufacturer to date, the core of the language has not changed much. The original 1954 version (similar to BASIC816) only supported **GOTO**, **GOSUB** / **RETURN**, and **IF** / **THEN** (line #).

Because of this rudimentary level of flow control (there was no **ELSE** command or compound logic), and because of the line numbering scheme, programs tended to be difficult to read and/or convoluted.

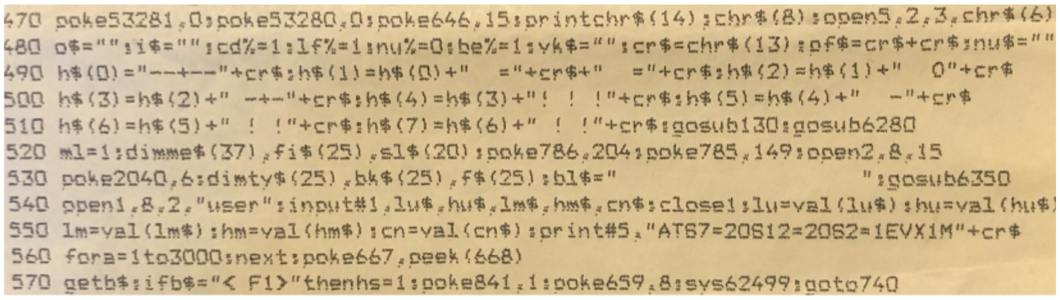
While doing research for this article, I was surprised to find that **GOSUB** / **RETURN** was part of the original language definition. This must have been revolutionary thinking at the time but then again, even vintage *processors* supported sub-procedures and assembly instructions for the basic three intrinsics (branch unconditional; branch on 'condition' such as overflow, borrow, zero; and jump subroutine (with return).

As an interpretive language, many BASIC versions store programs using a *token* methodology (and some, such as the early Sinclair machines, required you to choose the command to enter as a token via set of shifted keys which would token-ize immediately). Tokens reduce memory usage, improve execution speed, and limit parsing and error detection at run-time.

Commodore's PET BASIC introduced a full screen editor and a set of shifted shortcuts for statements that could be used to reduce the amount of typing and improve the amount of time required to enter a program.

A practical example of what not to do

Examine the code below. It looks old for a reason. It was written almost 40 years ago. No, that is not genuine parchment but 'yes', it was transferred from machine readable form to something called 'paper' with a then, relatively new invention called the dot matrix printer!



```
470 poke53281,0;poke53280,0;poke646,15;printchr$(14);chr$(8);open5,2,3,chr$(6)
480 o$="":i$="":cd%=1:lf%=1:nu%=0:be%=1:vk$="":cr$=chr$(13):pf$=cr$+cr$:nu$=""
490 h$(0)="--+--"+cr$:h$(1)=h$(0)+"   ="+cr$+"   ="+cr$:h$(2)=h$(1)+"   O"+cr$
500 h$(3)=h$(2)+" -+-"+cr$:h$(4)=h$(3)+"! ! !"+cr$:h$(5)=h$(4)+"   -"+cr$
510 h$(6)=h$(5)+" ! !"+cr$:h$(7)=h$(6)+" ! !"+cr$:gosub130:gosub6280
520 ml=1:dimme$(37),fi$(25),sl$(20):poke786,204:poke785,149:open2,8,15
530 poke2040,6:dimty$(25),bk$(25),f$(25):bl$="                         ":gosub6350
540 open1,8,2,"user":input#1,lu$,hu$,lm$,hm$,cn$:close1:lu=val(lu$):hu=val(hu$)
550 lm=val(lm$):hm=val(hm$):cn=val(cn$):print#5,"AT67=20S12=20S2=1EVX1M"+cr$
560 fora=1to3000:next:poke667,peek(668)
570 getb$:ifb$="< F1>"thenhs=1:poke841,1:poke659,8:sys62499:goto740
```

I wrote this in 1984 and broke all three of the five cardinal rules of BASIC programming.

- Don't **pack** unrelated code onto a single line (line 470 is 77 characters long and line 540 is 78!)
- Minimize the use of direct memory access (**PEEK**, **POKE**, and **SYS**) commands, if you can help it.
- Use spaces and **REM** statements to make code readable and understandable so another user or programmer can maintain it (or learn from it).

Here's a quiz: what does this program do?

Hard to tell, but this is from a BBS that I wrote and operated while in College in Berkeley California. Most interesting (as I read this today) was use of the `h$(x)` array which built 7 strings that when printed, displayed ASCII hangman for one of the many online games.

Now that we've illustrated what *not* to do, let's talk about some best practices in the context of this month's program.

In order to get everything into 25 lines, we've had to combine multiple statements onto a line, contradicting cardinal rule #1 above.

Line 610 is probably the most egregious example of this and line 100 is rather long as well.

But Line 100 is passable; it merely contains a series of simple variable definition statements (constants on the left and variables on the right, in this case). It's also executed only once, and is easy to modify if we need to tweak a setting. It's always a good idea to define all variables early on. As bad as my old code (above) was, I adopted this convention on line 480.

There are no **REM** statements in this month's program (as stored on the Foenix Marketplace) but we've inserted some in the code on the right (pg. 27). An interesting note about inline documentation; depending on the version of BASIC, remarks can actually slow down execution, but for programs that are not speed critical, remarks are recommended.

The line numbering scheme that we used here is intentional chosen in order to 'block' or group related code within a given range, as if subroutines. This serves two purposes: a) it leaves room for modification and adding new lines and b) it makes it more readable, assuming you've done this consistently and sensibly.

You might have noticed that the line numbering on the BBS example on page 25 was at a strict 10 number interval. The BASIC I used at the time had a handy **renumber** command which does not exist in BASIC816. But SuperBASIC, Simons' BASIC, and later versions of Commodore Basic (7.0) included such functions and they took care of all referenced line numbers. My BBS program was approximately 900 lines long; renumbering by hand would be untenable.

With regard to line numbering, if you want to see another example of what *not* to do, look back to issue #1's Commodore Balloon example on page 8 and the footnote pointing out my sense of displeasure.

Ok, back to the Foenix Ballon program. A keen eye will see that this program uses no **GOSUB** / **RETURN** pairs and instead relies on favoring jumping only when necessary. Notice that we order code blocks that 'fall' to subsequent code blocks to satisfy instances that either occur most, or benefit from higher performance. This is a technique used in assembly language as well, favoring falling-through versus branching and opting for instructions which require fewer cycles (such as relative branching) versus long jumps or `JSR` / `RTS` pairs that put pressure on the stack.

In our case, we are favoring balloon decent and movement from left to right; it makes logical sense since gravity pulling is stronger than a hot air balloon trying to escape it.

Code Description

Now let's move on to describe each code section. If you've printed this on double sided paper, the program listing should be directly across from this page. If not, I'm sorry :)

Line 100 is executed once and merely sets a bunch of variables.

Line 110 is executed once at the start of the program and then anytime the balloon hits the top of the screen. It resets a number of variables and re-points the sprite to the base image (no flame, descending).

Lines 120 - 140 are executed every iteration; they test to see if it's time to toggle the beacon and if so, jumps to that routine which jumps back; it positions the sprite at the 'new' *x* and *y* value, and it runs a tight delay loop for *gr* iterations. This is our gravity variable; note that it is a float, not an integer.

Lines 150 - 160 deal with vertical (*y*) motion. If ascending, then jump to 300; if not, simply increase *y%* by 1 pixel, subtract 0.2 from gravity, and test to see if we are at bottom; if so, goto 600

Line 170 checks to see if we are within the 'windy' zone by generating a random number * the current gravity value + 10 versus 75 and if less, it skips the horizontal movement and jumps back to 130. This is the only time we avoid horizontal movement and the effect of this skittishness is convincing since it's a) random and b) occurs more frequently at lower 'altitudes'. Since we haven't adjusted *x*, there is no need to visit line 120; we always do the *y* movement before getting to *x*.

Lines 200 - 270 contain the *x* block logic. First, we test to see if the balloon is coming 'in' (from sea, I suppose) or heading out, relative to the left edge of the screen; if heading in, we jump to 230. Otherwise, 210 is executed which adds 1 to *x%* and checks to see if we are at the right edge of the screen; if so, we reverse the *in%* flag and jump back to 120. Otherwise, we do the opposite which is to subtract 1 from *x%* and test to see if at the left edge. Note that *x* movement is only concerned about left and right edges and it's only complication is the *x%* variable controls beacon flashing indirectly (tested at line 120). *y%* motion, on the other hand, is much more tricky since it has to deal with swapping of sprite frames for flame animation and landing. Note that the default (ideal) state is to have the balloon descending and heading 'out' (increasing *x*). It's considered *ideal* because it is the aforementioned 'fall-through' code; there are fewer statements.

Lines 300 and 310 are used when the sprite is ascending; line 150 above brought us here because *as%* was = 1. Since we are ascending, we subtract 1 from *y%*; we also punch a new value into gravity that reduces it; you can think of this as the balloon accelerating or the 'pull' of the earth becoming less strong at higher altitudes. We test for balloon 'top' (aka *tip%)* here as well and if so, jump to 110; this is an uncommon occurrence happening only once every several hundred pixels of movement (*y%* moving from *tip%* + 1 to the bottom of screen and back to the top of the screen with all of the *x* movement between, animations, and all of the *gr* delays).

Line 110 is 'reset time', and essentially restarts the program, albeit in a different *x* (horizontal) position.

There is something worth discussing at line 310. We are here because we are ascending and have not yet reached the top. So 4 out of 5 times (based on the MOD function), we go to the *x* motion block; on the 5th occurrence, we fall through to line 400.

Lines 400 - 420 flip to the next frame of the flame animation sequence by executing the SPRITE command with the new sprite address based on the *spr%* * 1024.

Finally we test to see if we are still within range of sprites (< 11). If so, we add one to *spr%* for next time and jump to the *x* motion block. If we were at 11, we reset *spr%* to 5 (knowing that the code will increase it by 1, which is what we want) and do something fancy to color #1 (which is usually black).

In this case, we use some calculations to plug in 1/2 the value of *y%* in the RED color, always use 0 for GREEN, and plug a random color between 1 and 250 into BLUE. We also increase the sprite frame # and goto the *x* block at line 200.

Lines 500 - 520 is the toggle beacon code and is similar to lines 1020 - 1050 on page 23 of last month's column.

Lines 600 - 610 is my favorite section. The balloon has landed, so switch the *as%* flag to 1, and run the animation delay loop on 610.

The FOR / NEXT loop serves two purposes: a) it's just enough of a delay between frames to be convincing and b) this single statement within the loop calculates the new address and does so within a SPRITE command.

SPRITE 0, 0, &hB00000 + INT (de% / 100) * 1024

The fact that we have access to functions like INT and can divide a variable within parenthesis inside of the function is useful; BASIC is a primitive language and does not always afford such luxuries.

Think about our proposed DIM$ array approach discussed on page 24 last month. That would have taken a dozen lines of code. With the aid of 12K of sprites, we traded memory for simplicity. It's nice to have choices.

Program Listing - be sure to first execute the pre-requisite / setup commands detailed on page 28 below

```
100 tip%=64:left%=58:bot%=448:right%=614:x%=58:y%=64:in%=0:be%=0:spr%=6
110 gr=80: as%=0: SPRITE 0, 0, &hB00000: SETCOLOR 0, 1, 0, 0, 0

120 IF x% MOD 40 = 0 THEN 500                     : REM if x divisible by 40, do beacon
130 SPRITEAT 0, x%, y%
140 FOR n%=1 TO INT(gr): NEXT

150 IF as% = 1 THEN 300                           : REM vertical direction block
160 y%=y%+1: gr= gr - 0.2: IF y%=bot% THEN 600    : REM descending
170 IF INT(RND()*10+1)*gr+10 < 75 THEN 130

200 IF in% = 1 THEN 230                           : REM horizontal direction block
210 x%=x%+1: IF x% = right% THEN 270              : REM   if moving right
220 GOTO 120

230 x%=x%-1: IF x% = left% THEN 260               : REM   if moving left
240 GOTO 120
260 in%=0: GOTO 120
270 in%=1: GOTO 120

300 y%=y%-1: gr= (y%/10): IF y% = tip% THEN 110   : REM ascending
310 IF y% MOD 5 <> 0 THEN 200

400 SPRITE 0, 0, &hB00000+(spr%*1024): IF spr% < 11 THEN 420
410 spr%=5: SETCOLOR 0, 1, y%/2, 0, INT(RND()*250)+1
420 spr%=spr%+1: GOTO 200

500 IF be% = 1 THEN 520                           : REM toggle beacon color
510 SETCOLOR 0, 73, 125, 164, 45: be%=1: GOTO 130
520 SETCOLOR 0, 73, 130, 33, 29 : be%=0: GOTO 130

600 as%=1:FOR de%=100 TO 600                      : REM landing animation and launch
610 SPRITE0,0,&hB00000+INT(de%/100)*1024:NEXT: FOR de% = 1 TO 800:NEXT:GOTO 120
```

Available on the Foenix Marketplace as "I03P27a.BAS" (click here)

How to leverage `.PAL` files within your program - pre-req to the main program

In prior articles, we 'setup' the sprite with a stand-alone BASIC program that loads sprite data into memory, does a MEMCOPY, and instantiates a palette using a few dozen DATA statements and SETCOLOR. Here, we will accomplish the same task with the `.PAL` and a single SPRITESHOW command. These 5 lines are prerequisite to running the program above and much more efficient than the approach we used in issues #1 and #2.

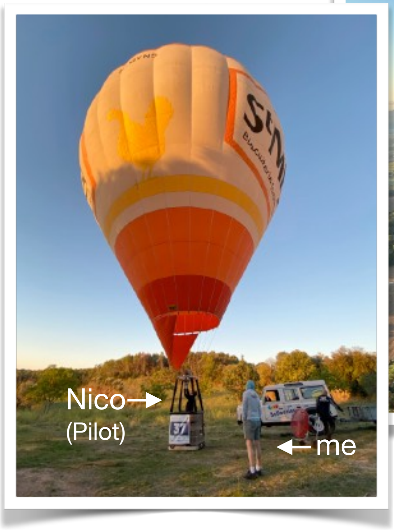
BLOAD "FBALLOO5.SPR", &h100000-1 (note the *minus one* to skip 1 byte of metadata and align sprite data to 1MB boundary)
MEMCOPY LINEAR &h100000, 12288 TO LINEAR &hB00000, 12288 (12 sprites * 1024 = 12,288 bytes)
BLOAD "SPREDTV1.PAL", &h100000 (we load the palette into the same space as above since the sprites are in VRAM by now)
MEMCOPY LINEAR &h100000, 1024 TO LINEAR &hAF2000, 1024 (move palette data; 1024 bytes; see pg. 22)
GRAPHICS 47: SPRITESHOW 0, 1 (set graphics mode and turn sprite #0 'on'; lines 110 and 130 take care of the rest)

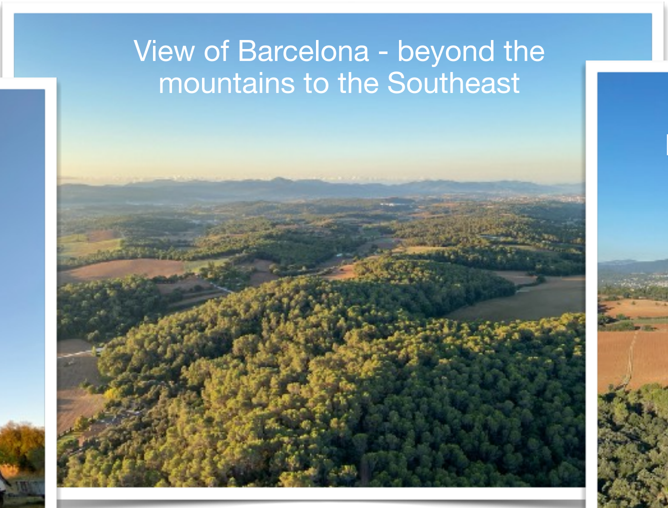
Beginner's Corner says farewell for now…

Now that you are equipped with tools and examples to experiment with, I'd like to ask a question and a favor. Do **you** have the get-up-and-go to try some of this for yourself. If so, I'd be interested in your results and learnings and also invite you to write an article for a future issue of this newsletter and/or to share your work on Discord.

Of course this is a simple example and not quite Space Invaders, but it could be adapted to be something like it. (the original Space Invaders had 55 aliens, one player, one player missile outstanding, one part time UFO, and 3 or 4 alien missiles onscreen at any one time. Easily doable on a 64 sprite Foenix system).

On a related note, here are a few pics from a hot air balloon ride that my wife and I took, traversing 3,000 ft. elevation across the Catalonian countryside, in early October. I need to share that landing is far more nerve racking than flying; our pilot (Nico), a Frenchman and competition flyer, had over 2,000 flights under his belt, so that helped!





Next for this column and for Foenix Rising in general

Starting next month, I'll be doing more *actual* development and spending less time writing and editing text for Foenix Rising. Therefore, I am asking the community for thoughts and comments about this newsletter and most importantly, for contributions (content) so we can carry this endeavor forward. Following this issue, I will be publishing one more issue at year-end and then moving to a quarterly format. Beginner's Corner will be back in 2023 with something new!

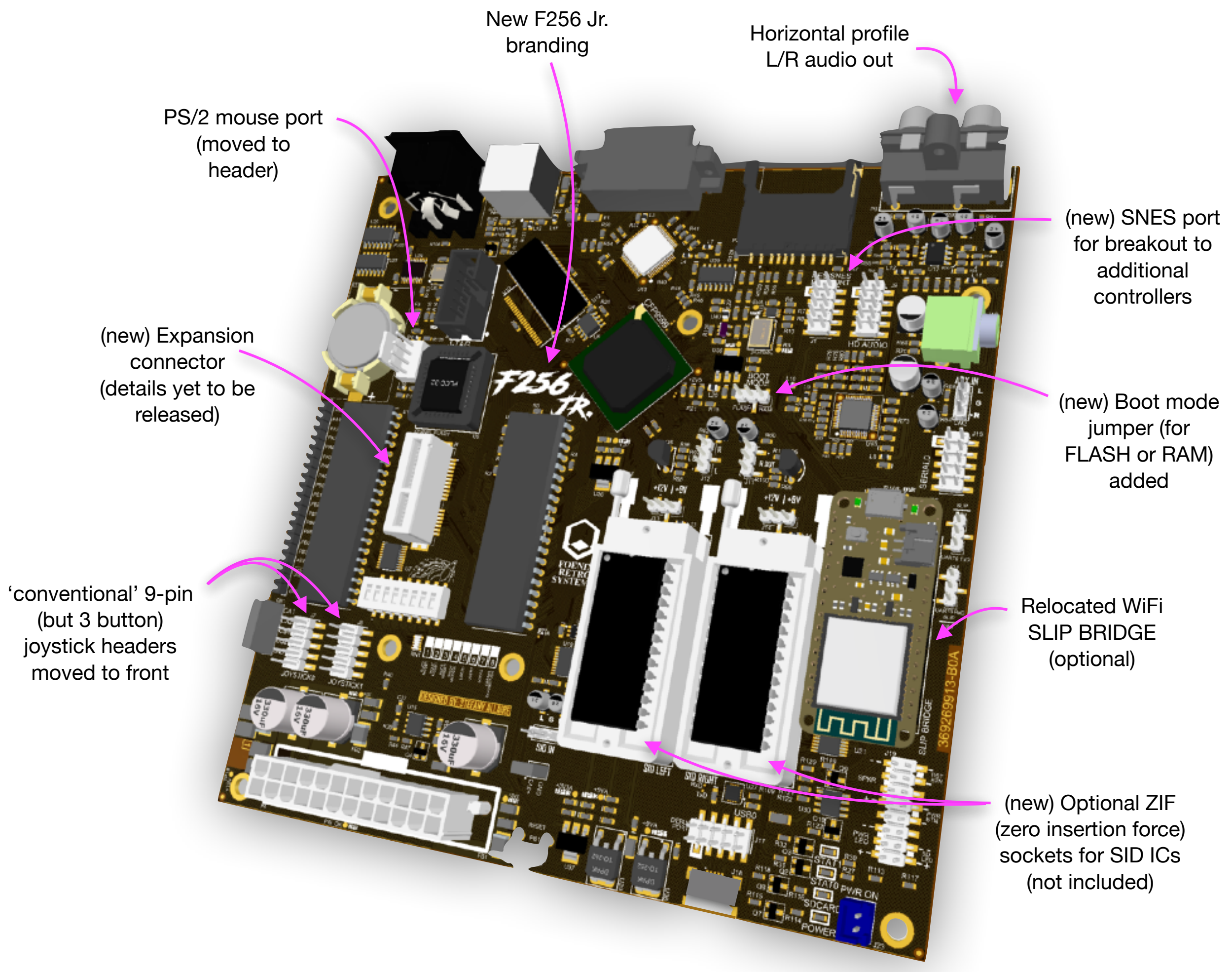
For me, I have three software development efforts that I will be starting later this month:

- The *Gran Trak Foenix* project mentioned on page 5 above
- A port of a Commodore 64 game developed by a 6502 hobbyist, based on Taito Space Invaders. I'll be porting it to 65816 / C256 Foenix, and the F256 Jr., primarily as an academic exercise.
- A Unix *curses* based wrapper for vintage text based applications for the the A2560K, threatened previously

F256 Jr.

Details of the rebranded rev. 'B' released

The C256 Jr. dev board has transformed into the F256 Jr. product. The new revision is lower in profile and has several additional hardware features. Best of all, software is evolving at a rapid rate. This month, we'll do a quick fly-over and then talk power and wiring (applicable to both Jr. boards)

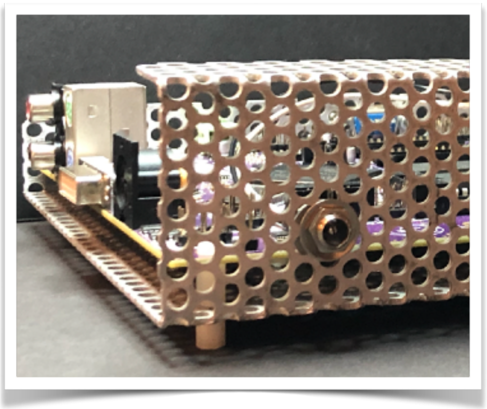

New F256 Jr. branding
Horizontal profile L/R audio out
PS/2 mouse port (moved to header)
(new) SNES port for breakout to additional controllers
(new) Expansion connector (details yet to be released)
(new) Boot mode jumper (for FLASH or RAM) added
'conventional' 9-pin (but 3 button) joystick headers moved to front
Relocated WiFi SLIP BRIDGE (optional)
(new) Optional ZIF (zero insertion force) sockets for SID ICs (not included)
F256 JR.

The diagram above calls out noteworthy changes and new features since the revision 'A' development board.

What remains the same is the form factor (although this design is lower profile). If you missed the feature on the C256 Jr. revision 'A' dev. board, have a look at Foenix Rising issue #2 (August), pgs. 26 - 28. With the exception of the PS/2 mouse port being moved to the 4 pin MTA connector, every jumper and feature of the dev. board applies to the production version, including the CPU footprint option.

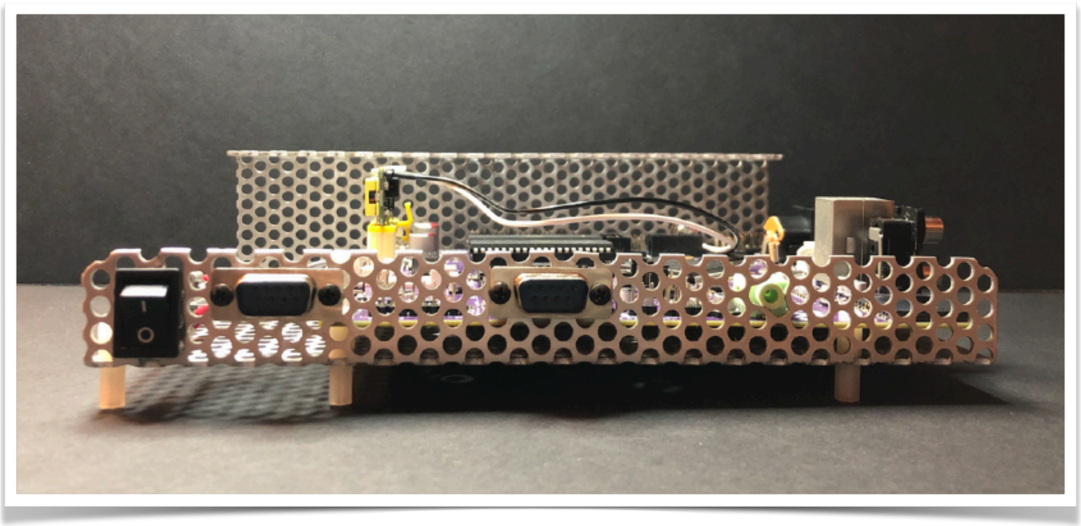
Elsewhere, there has been significant progress on FPGA features, on Gadget's kernel development efforts, and a powerful new BASIC language called *SuperBasic*, written by Paul Scott Robson. A comprehensive manual, written by Peter Weingartner (author of the A2560K MCP kernel) is also in the works. Download a work-in-progress draft here. (see the `github` / resources section on pg. two for links to repos, otherwise)

The following pages discuss wiring in the context of my own industrial style mini-ITX case. See the 'contest' note in the puzzle section on page 16-17 for a chance to win a custom made case just like it!

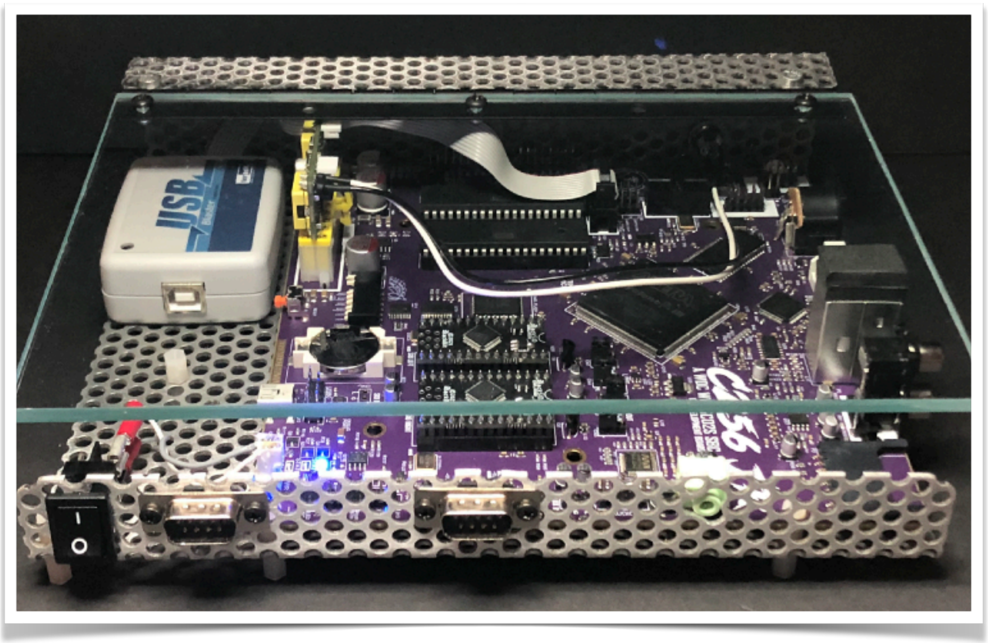


Cheese grater - many of the smaller ITX, and especially “picoPSU” supplies, come pre-wired with a barrel connector. Feed it 12V DC from a capable wall wart or modest bench supply and it will grace your system board and light peripherals with power and signaling required to get off the ground quickly.

In my prototype case design, the back panel is barren, with only a power jack present.



Scorpion - without the lucite cover, the profile of this hand-braked case resembles the predatory arachnid. I opted for a hard on/off switch and nothing but dual joysticks and the headphone jack up front. All of the onboard ports exit to the right where it’s easy enough to get to, and in such a way that does not clutter the precious space (a pet peeve) between my Jr. and monitor.

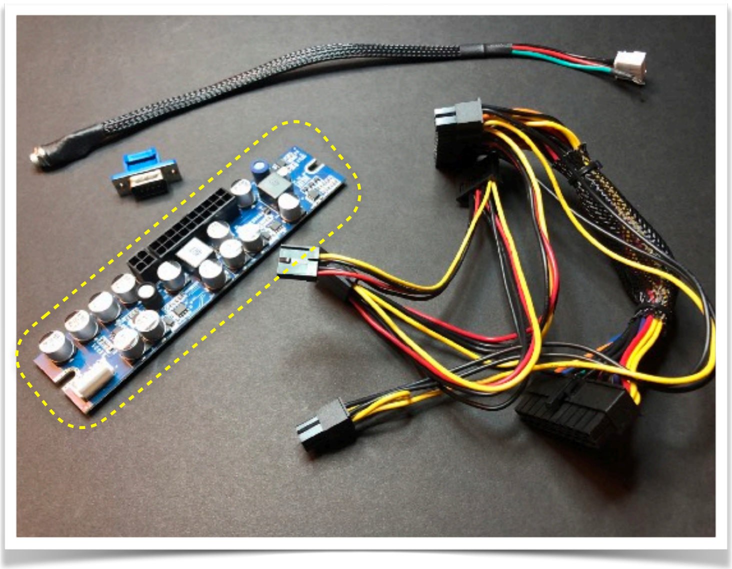


“Hinged roof, covered” - in this pic, joystick ports (front) are not yet wired but the JTAG interface is connected. The left side is open for easy access to the USB debug port and the *terasIC* FPGA blaster.

Wiring a hard power switch requires only two short leads of wire from the switch to the MTA connector. There is also a header between the joystick ports that can be wired to a momentary switch if that is more your style.

Power options (a few to speak of)

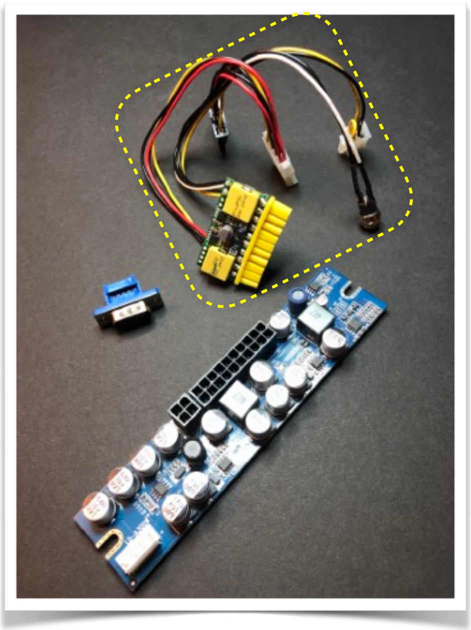
NO! Resist the temptation to use an open frame industrial power supply. ATX / ITX power is multi-voltage so you will need something a bit 'more' robust than the garden variety 12V DC adapter. The options detailed below are inexpensive and easy to procure. Power technology has come a long way.



The LD A300W - Amazon [currently] $29.99 USD Sold by "RGEEK". Comes with the cables shown (DB9 pin connector added to the picture for scale).

I do *not* recommend this for the Jr. It's overkill. This is an ideal power supply (well rated) if you have one or more peripherals to power (SSD, floppy or spinning disk). I believe the GEN X uses something like it.

The cable braid (while it may be modified to be a bit lighter) is too stiff and will clutter your case for no good reason.



PicoPSU 90 - Amazon [currently] $29.95 USD - sold by "MITXPC" Comes with the cables attached (I left the LD A300W in the picture to show the size difference). This is what I used in my case.

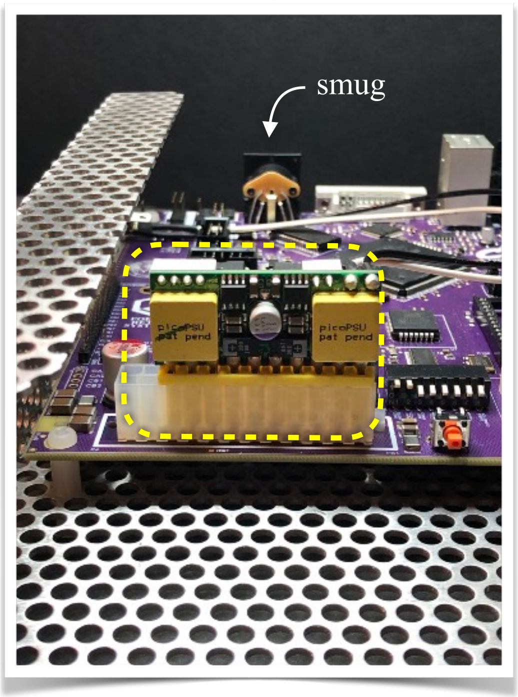
The peripheral power cables are removable! All you really need is the barrel connector which is hardwired (soldered) to the board.

Word of caution; these are commodity products so read the reviews carefully and don't be shocked if the item received is slightly different than the picture you saw when you placed your order!

Pictures on the subsequent pages show a closer look at this power supply and the wiring. Note that this has a **20 pin** connector (2 rows of 10). The Jr. board will accommodate either 20 or 24 pin.

Most ITX power supplies require separately purchased AC adapters of *reasonable* power. While a wall wart style will work, I chose to invest in one of the Harbor Freight / Amazon sourced "Pyramid" supplies. They are switched, can be easily serviced (recapped), and have pots to adjust voltage.

(also, you can use banana plugs which are fun and confuse friends and family) (right) Closeup of the picoPSU. The only wiring required is the pre-soldered black and white cables on the upper-right which loop around, then back to the far end where the barrel connector is mounted (see below, also). Notice the IEC connector in the background turning its back to the camera. (how rude!)

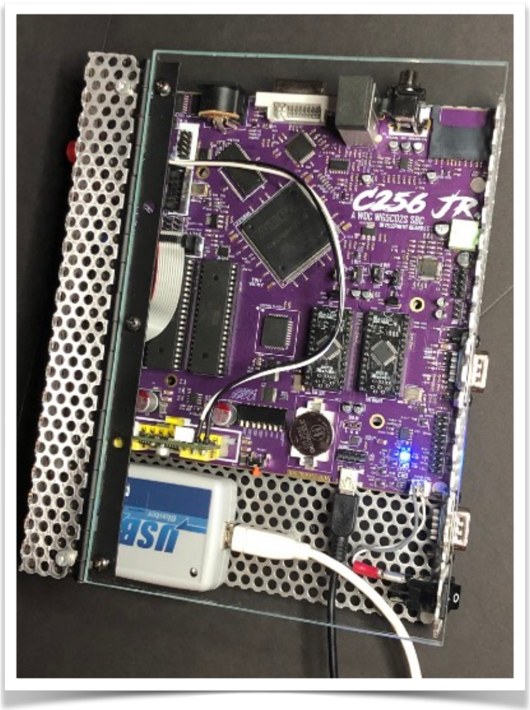


A real "page-turner" :)

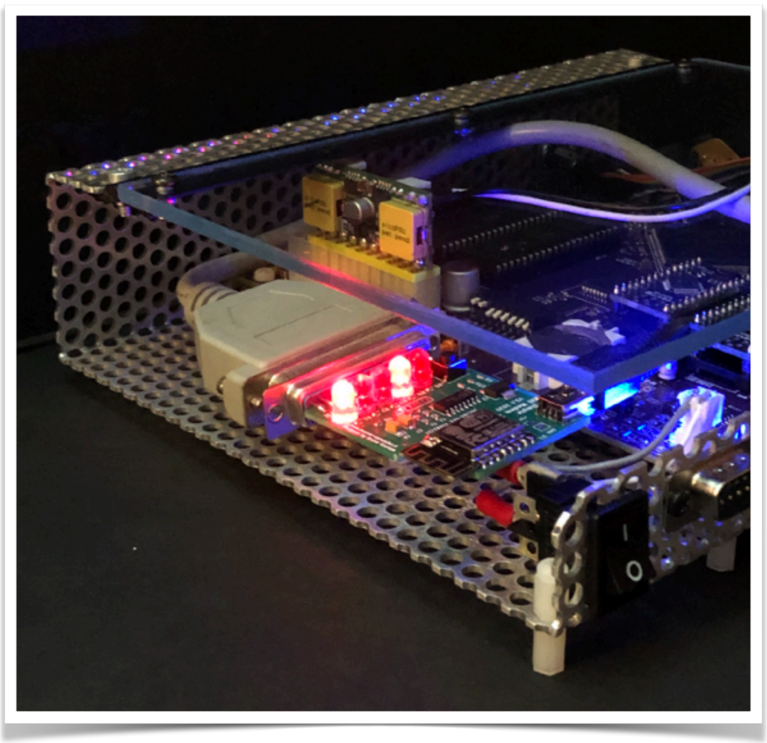
(right) Simplified view to clearly show the internal power cable and on/off switch wiring.

In this pic, you can see the USB debug cable (black) and USB JTAG cable.

Monitor and keyboard are not connected and the joystick ports are not wired in this picture.



Invest in tools and learn skills like our ancestors did !!



(left) Here, the serial header is wired to a db9-to-db25 to provide the plumbing for a USB powered WiFi modem for some sweet 9600 baud BBS surfing.

One of the benefits to designing your own enclosure is to accommodate off the beaten path features. Most of the store-bought cases are small footprint but have an excess of height, which I find cumbersome.

As depicted on pg. 29 and in last month's issue, the Jr. has all sorts of headers and accommodations for external status LEDs and audio and DAC action, not to mention the VIA driven 20 pin Commodore keyboard connector. #MAKEITYOUROWN