SHIFT838 Newsletter

JUNE 14, 2015 VOLUME 1, NO 6

This newsletter is dedicated to the ongoing support for the Texas Instruments 99/4A and Myarc Geneve 9640 user community and is published by SHIFT838.

In this edition of the newsletter the interview I was able to obtain is with Matthew Hagerty, designer of the F18A video upgrade for the TI-99/4A as well as other older systems.

I was also able to get a coding example of file IO for use with Turbo Forth from Mark Wills. This was very much needed and I am happy to be able to publish it as it will allow a lot of as, myself included to better understand File IO with TurboForth.

Thanks to all that have subscribed!

Interview

Matthew Hagerty

Q: When was the original idea thought of for the F18A and what made you think of it?

A: *Back in 2005 or so I attended the TI Faire in Chicago for the first time. In the retro-computer scene Jeri Ellsworth had just done the C-One (C64 SoC) and during a break-out session the question of FPGAs came up. That got me thinking about FPGAs, but I figured someone else would implement SoCs of all the classic home computers very quickly, following Jeri's lead. The idea sat in the back of my mind and in 2008 I bought an FPGA devboard, which promptly went on a shelf until around 2011.*

Sometime in 2011 there was probably yet another thread on the forum about someone trying to get better video output from the 99/4A, either by replacing the 9918A with a 9928 to get component video, or by external analog conversion of the 9918A's composite signal. Such projects have never seemed to produce enough improvement to be worth the effort or required modifications (soldering, cutting traces, etc) to the 99/4A that would prevent many users from being able to get the upgrade. So I started thinking about where the video would be the best quality, which of course is inside the 9918A as raw data before any output video generation. I have always been fascinated with computer graphics and I always wanted to know how systems like coin-op video games generate video because they don't use a single VDP chip like the 9918A. I figured I would try re-implementing the 9918A using an FPGA which would allow me to learn all about these circuits, as well as learning an HDL

(hardware description language), digital architecture, FPGA to external circuit interfacing, circuit board design, layout, and manufacturing.

In the back of my mind I always thought it would be nice to be able to make a simple drop-in replacement for the 9918A that users could easily plug-in with minimal skills (taking the console apart), but I had to make the thing first, and it had to work.

Q: Why is the F18A compatible with so many systems?

A: *The F18A is compatible with so many systems because the original 9918A/9928/9929 VDPs are general purpose video chips and are used in so many systems. The F18A is designed with the same physical pin configuration, electrical specifications, and interface timing of the original VDPs. The F18A circuit board is a little wider than the original 40-pin DIP package (which I'm not happy about), but other than that it can directly replace the original VDPs.*

Q: What elements of the 9918 were the most difficult to implement?

A: *The most difficult element of the original VDP is the one I did not implement: the color composite video output. Even though the F18A's primary goal was to output a clean high-resolution VGA video signal to drive modern monitors, I did also want to try and output the original composite video as well. Needless to say, VGA output is easy, composite NTSC/PAL color video is not. I ran out of time, money, FPGA resources, and physical board space trying to get composite output added in.*

Internally the most difficult elements were those that are vaguely documented or not documented at all. For example the 5th sprite number in the status register, the actual frame interrupt trigger (it can vary by a few scan lines, but it is not known exactly why), etc. This type of functionality is known only through characterization of the original VDP, and digging up that kind of information takes a lot of time, research, and testing.

*Also, while I was still learning digital circuit design, VHDL, and FPGAs in general, just figuring out how to implement certain functionality was difficult. I got to a point where I was trying to add sprites and the circuits just kept failing. It was at that point that I took some time, bought another round of books, and stopped thinking in terms of programming. Since I was coming from a predominately programming background, and HDL syntax *looks* like programming, you can get into trouble very quickly with a software mindset. Transitioning to a hardware mindset was a game-changer for me and allowed me to finish the design and improve the circuits.*

Q: There are obviously several restrictions/limitations of the old hardware that could have easily been wiped out in the new FPGA architecture... What kind of things did you need to keep in place to maintain backwards-compatibility with existing software?

A: *Sprites are probably the best example of this. Being able to display all 32 sprites on a line at one time is possible with the F18A and I originally thought "No more flicker! Who would not want that??" However, while most software suffered from the four sprites per line limitation, I found out the hard way that some software relies on it. In some software the fact that only four sprites will be displayed is used to mask other sprites, detect the horizontal scan line, etc. The default number of sprites on a*

line (4 vs 32) was responsible for the addition of the physical user-header (jumpers) on the F18A circuit board. Being able to display all 32 sprites, but keep original functionality / compatibility is something that I have worked on all the way up to the latest V1.6 firmware.

Implementing and sticking to only the original functionality would be much easier. As soon as you start adding features and expanding the design you have to be very careful since seemingly unrelated changes can affect existing software (and hardware!) in ways you don't realize.

Q: Following the initial release and subsequent feedback from users across multiple systems, what were some suggestions you received on bugs or requested features?

A: *Bugs came first. I believe the first reported problem was the XB Penguin game where sprite collisions were detected incorrectly. This blew me away that an XB game, of all things, exposed a problem. As it turns out, the F18A was detecting sprite collisions off-screen (which the 9918A datasheet leads you to believe happens) but the real 9918A will only detect collisions for sprite pixels on the active display, i.e. in the 256x192 pixel area. Using CALL COINC(ALL) in XB exposed the bug because on the F18A the off-screen sprites were causing the collision bit in the VDP status register to be set.*

There were a few other bug reports which lead to the V1.4 and V1.5 firmware. Firmware updates are a problem because not everyone installs the update, and right now the only in-system updater available is for the 99/4A. Originally I had planned on including an on-board USB interface for updates, but again time, money, and board space forced me to cut this feature.

The first feature request that I could do something about, i.e. a request other than more VRAM, HDMI output, cheaper, or to redesign the board, was from Rasmus.

Q: How much did user feedback play in to the new released firmware update?

A: *Feedback and requests were a huge part of the V1.6 firmware, as well as additional changes to support the little known 9118A/9128/9129 VDPs that were short lived and came out after the 9918A/9928/9929 that we are most familiar with.*

The V1.6 changes started when Rasmus tried using some of the original "features" and we realized they were pretty much useless. That is a problem when you are developing features without context, you have no reference if a feature will be useful or not. I started making changes based on feedback and the more I changed the more Rasmus asked "hey, this seems easy, can you add this?" and "it would be cool if we could do ..." The V1.6 changes started in April 2014 and I took a break during that time to change jobs, but all-in-all it represents 6 to 8 months of work, a totally rewritten tile generation subsystem, new and useful features, etc.

If not for feedback and suggestions, V1.6 would probably have just fixed a few final bugs. Instead the V1.6 firmware includes massive changes and tons of new features.

Q: Could you briefly describe some of the more interesting features of the new firmware update?

A: *To me one of the most exciting and interesting features available in the new V1.6 firmware is the second tile layer which lets you put a second tile layer on top of the first tile layer. Both tile layers are completely independent and can be pixel scrolled (both horizontal and vertical) separately, and both have their own name table and color table. Rasmus made a fantastic set of demo programs for the V1.6 firmware, and the 2-plane demo shows off the new tile layer capability beautifully.*

Another interesting feature is the alternate tile attribute capability added for the text modes (T40 and T80). This allows you to assign the foreground and background color for each tile independently. This is a great feature for terminal emulators since they can now display true ANSI colors on BBS systems. This also allows enhanced text editors to display highlighted text, color coding, and other such features.

Some other features worth mentioning are better use of VRAM for the enhance color modes (more colors per pixel), scrolling support for the text modes, a very high precision timer, GPU hsync and vsync triggers, and more.

Follow up:

Q: How do you see application and development progressing--going forward-- given the new capabilities of our old system (thanks to the F18A)

A: *I'm not sure how the F18A will affect development going forward. If history is any indicator of the future, unfortunately it is looking pretty bleak. I'm certainly not discounting the great programs that have been developed that use some F18A features, but the F18A has been out since the summer of 2012 and there is still no new software that is written for the F18A only.*

All the applications that support F18A features also support the original 9918A VDP, and there is currently a big interest in pushing the original VDP and doing things we previously though impossible. I think it is very cool to see such things on the original VDP, and had such examples existed before I started the F18A I probably would not have implemented pixel scrolling or some other features.

None of this is not a complaint from me, I understand people writing software want to make it available to the widest audience, which is currently people without an F18A. However, this also suggests that software development won't change much going forward.

Q: What are some of the advancements that you foresee for the F18A in the future with the 99/4A

A: *I'm not sure what is meant by advancements, but the F18A is a fixed piece of hardware so advancements would have to be in the realm of bug fixes for the most part. To really squeeze more functionality out of the existing device would require rewriting most of the circuit descriptions.*

Q: Have you thought of possibly releasing a Software Development Kit for the F18A that would allow users to utilize the F18A features for programming environments? Such as the below:

A: *Absolutely, I have plans for coding examples, documentation, etc., all I need is another life-time. I have a family and a day job, so hobbies always take a back seat to those.*

- **Extended Basic programming to have a program that could be load into high memory utilizing CALL LINKS**

  A: *I'm not sure about providing much for XB. The biggest problem with XB is staying out of the way of XB. It is everywhere in the system and so easy to step on its toes and corrupt the environment. Many of the great F18A features require control and use of VRAM which would cause problems for XB.*

- **Assembly for Editor Assembler DIS/FIX 80 (option #3 [Option 4 on XB2.7 suite]) and PROGRAM image files (Option #5 [Option 6 on XB 2.7 suite]) programs**

  A: *Certainly, assembly is currently the best environment to take advantage of the F18A. I'm not sure what I might provide as far as tools in the future, but example and documentation are on my list.*

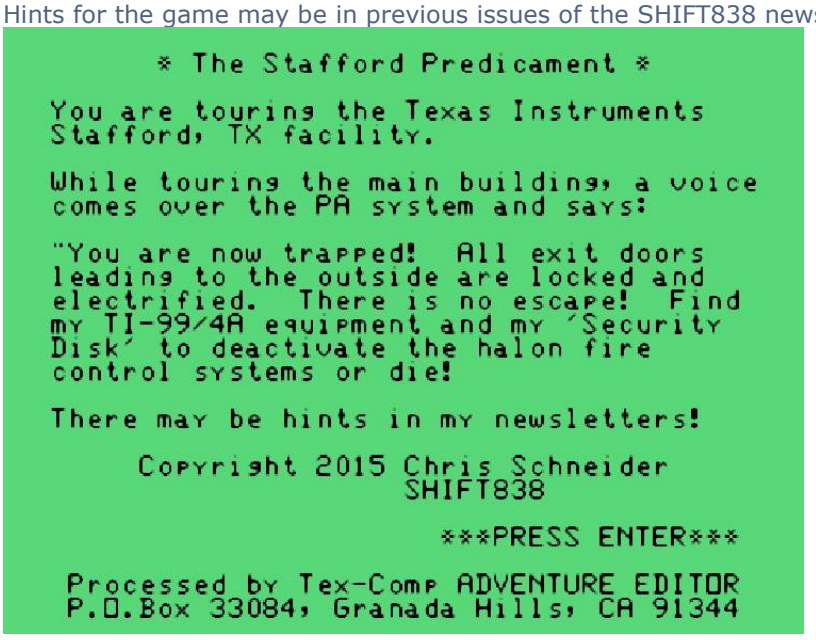
- **TurboForth**

  A: *I'm not sure what I can do for TF programmers other than support Willsy in implementing F18A support into TF. TF currently supports T80 on the F18A, but I'm not sure what else is planned. If TF allows direct access to the VDP registers and makes no assumptions about how VRAM is used, then a TF programmer has everything they need to take full advantage of the F18A.*

Software

The Stafford Predicament – This is an adventure game used with the Adventure module. It utilizes 97% of available memory and is very extensive. By far the largest adventure I have ever coded. I spent more than 100 hours coding this. There is an Easter egg in the adventure for you to see if you can discover (**no sector editors please!**).

Hints for the game may be in previous issues of the SHIFT838 newsletters!



The adventure has been uploaded to the ftp site, Heatwave BBS and Atari Age, as well as you can download it directly from my site.

http://shift838.wix.com/shift838#!ti-994a-software/cuog

ftp://ftp.whtech.com/Users/Chris_Schneider/Adventures/SH838-ADV.dsk

Coding

I wanted to touch on some basic IO access via TurboForth to continue from last months newsletter. I have been in contact with Mark Wills to get an example of some basic File IO. Mark put the below together for publishing in this months newsletter. Thanks Mark, this will help a lot of us have a better understanding of TurboForth File IO.

```
       1        2        3        4        5        6        7
.........|.........|.........|.........|.........|.........|.........|
```

Opening Text Files in TurboForth

Chris recently asked me how simple file I/O is done in TurboForth, so here is a quick article to show how to read a text file (DV80) from disk.

If you have any experience with file I/O in TI Basic or Extended Basic then you will find that getting ready tp open a file in particular is very different in TurboForth. However, once the file has been opened, it's all quite natural; not that different from Basic.

The code shows the procedure. Everything you need to know/do to open a DV80 file for input. The code is heavily commented. A pure versionof the code, with all the comments removed is given later.

If you don't know anything about Forth, the the best place to start is here:
http://turboforth.net/about_forth.html

The above link builds on very first principles. Then, when you have read that, have a look at the tutorials on TurboForth.net:

http://turboforth.net/tutorials/tutorials.html

The tutorials start from the very first basic principles and build up in complexity.

So, here's the code to open a DV80 file and display it:

Lines of text with a preceding **** are comments and are ignored by TurboForth...

```
\ first, create a file buffer. This is used to hold information
\ about the file(s) we want to work with.
\ using the word FBUF: we're going to create a file buffer called fileIn.
\ Once defined, fileIn exists as a word just like any other word. When
\ used in a program/word, it will push the address of its buffer to the
\ stack.
```

```
FBUF: fileIn

\ now, create a buffer to hold the lines of text/data that is read in
\ from the file. Here, we create a buffer called buffer, with a size
\ of 82 bytes (chars).
CREATE buffer 82 CHARS ALLOT

\ now define a word called typeFile. This is where all the work
\ will be done...
: typeFile ( -- )
  \ now we're going to use the word FILE to define the name of the file
  \ that we want to open, and its type. FILE needs two things:
  \ a string containing the file name and file characteristics, and the
  \ file buffer to use. As can be seen below, were giving a string and
  \ also the file buffer fileIn to FILE. FILE will then build a PAB
  \ (peripheral access block) in the buffer. All subsequent file ops on
  \ this file will use fileIn as a reference.
  S" DSK1.TEST.TXT DV80SI" fileIn FILE
  \ in the string above, DV80SI simply informs the file system that we
  \ want to open the file as *D*isplay *V*ariable 80, for *S*equential
  \ *I*nput. There are other modes. They are all described here:
  \ http://turboforth.net/lang_ref/view_word.asp?ID=265
  \ now, we can use #OPEN to physically open the file. Of course, #OPEN
  \ needs to know which file to open. Where is that? It's in the file
  \ buffer that we called fileIn - so we supply that to #OPEN
  fileIn #OPEN
  \ #OPEN pushes a value to the stack to tell us if the OPEN command
  \ worked or not. If the open *failed* then it pushes TRUE (-1) to the
  \ stack. If the open succeeds, then it pushes 0 (false) to the stack.
  \ we use that value from the stack in the following IF statement. Note
  \ that the IF *removes* the value on the stack that #OPEN placed there
  \ for us. So, if the value on the stack is TRUE the code inside the
  \ IF...THEN block will execute.
  IF
    ." Could not open the file."
    ABORT \ abort will return us back to the command line
  THEN

  BEGIN
  \ here, we use #EOF? to determine if we're at the end of the file or
  \ not. Which file? #EOF? needs to know which file we're talking about.
  \ again, that information is in the buffer fileIn.
    fileIn #EOF? NOT WHILE
  \ so, while the file is NOT at the end of the file, we use #GET to read
  \ a single line of text from the file. #GET needs to know which file
  \ to read from (fileIn) and it also needs to know where to put the line
  \ of text that it gets from the file. At the very top of the code we
  \ defined an 82 byte buffer, called buffer. Recall that specifying the
  \ name of the buffer causes it to push its address to the stack. This
  \ is used by #GET
      buffer fileIn #GET
  \ #GET pushes true to the stack if it could NOT read from the file.
  \ So, we use that value with IF and abort with an error message if the
  \ #GET failed.
```

```
      IF
        ." Could not read from file"
        ABORT
      THEN
  \ if we get to here, then #GET succeeded, and the line of text is in
  \ the buffer called buffer. The *first byte* of that buffer contains
  \ the length of the string that was read in from the file.
  \ COUNT takes the address of the buffer, reads the first byte, and
  \ pushes the length of the string, plus the address of the first byte
  \ of the string to the stack.
     buffer COUNT
  \ we then use TYPE (which requires the address of the string, and the
  \ length of the string (how convenient - COUNT just put them on the
  \ stack for us!) to type (print) the string to the screen. CR then does
  \ a carriage return, which moves the screen position down to the next
  \ line.
  TYPE CR
   \ having done that, we're ready to repeat the whole process. REPEAT
   \ causes a jump back up to the word BEGIN. Thus all words/code
   \ between BEGIN and REPEAT will be repeated. When #EOF? detects an
   \ end of file condition, then the code after WHILE will NOT execute
   \ and the program jumps to the code AFTER the word REPEAT.
   REPEAT
   \ when we read the end of the file the program flow jumps down to here.
   \ we simply use #CLOSE to ask it to close our file, and we're done.
   fileIn #CLOSE
   ." All done" CR
;
```

Phew!

The code, with all the comments removed, looks like this:

```
FBUF: fileIn
CREATE buffer 82 CHARS ALLOT
: typeFile ( -- )
  S" DSK1.TEST.TXT DV80SI" fileIn FILE
  fileIn #OPEN
  IF
    ." Could not open the file."
    ABORT \ abort will return us back to the command line
  THEN

  BEGIN
    buffer fileIn #GET
    IF
      ." Could not read from file"
      ABORT
    THEN
    buffer COUNT
    TYPE CR
   REPEAT

   fileIn #CLOSE
```

```
   ." All done" CR
;
```

The above code can be pasted straight into Classic99 and executed immediately. You'll need to give a name of an extant file on one of your disks, of course! Just type '**typefile**' and it will open the file and display it.

Remembrance of TI-99ers

For a list of TI-99ers no longer with us please visit the remembrance page: http://ti99ers.org/modules/Inspire/remember.htm

Resources

Contact information
To contact me please feel free to visit my website and click on the 'Contact' tab.

http://shift838.wix.com/shift838

Newsletter Topics
If you would like to participate in the writing of this newsletter or provide any topics for this newsletter please contact me via my web site.

Sites

There are a few of sites that I think should get their own list below. These are for the TI Hall of Fame and TI-99ers Unsung website. Please visit these below sites as both have great information.

http://www.ti99hof.org/index.html

http://www.ti99ers.org/unsung/

Also the below site has a list of all the TI-99ers that have passed.

http://ti99ers.org/modules/Inspire/remember.htm

Below resources are just a handful of sites that support the TI-99/4A and/or Geneve 9640 computers. It is in no way a full list. This section will be included in all future newsletters. If there is a site that you think should be mentioned then please contact me.

Web sites / FTP Sites

http://www.99er.net

http://www.ninerpedia.org/

ftp://ftp.whtech.com

http://shift838.wix.com/shift838

http://www.ti99-geek.nl/

http://www.mainbyte.com

http://www.atariage.com

http://www.harmlesslion.com

http://www.ti99iuc.it

http://www.turboforth.net

http://www.ninerpedia.org/

Yahoo List Groups:

https://groups.yahoo.com/neo/groups/TI99-4A/info

https://groups.yahoo.com/neo/groups/TI994A/info

https://groups.yahoo.com/neo/groups/Geneve9640/info

https://groups.yahoo.com/neo/groups/turboforth/info

https://groups.yahoo.com/neo/groups/swpb/info

TurboForth & fbForth Resources

AtariAge TI-99/4A forum:
http://ti99.atariage.com

Main fbForth thread with release stuff always up to date in the first post:
http://atariage.com/forums/topic/210660-fbforthti-forth-with-file-based-block-io-post-1-updated-12052014/

Main TurboForth resource:
http://www.turboforth.net

Both have resources listed here:

http://atariage.com/forums/topic/153704-ti-994a-development-resources/

Active BBS'

HeatWave BBS

Access: Dial-Up and Telnet
System: Geneve 9640
Software: S&T BBS Software
Location: Arizona
Content: TI and Geneve file libraries, message bases, door games and e-mail.
Telnet to: www.heatwavebbs.com port 9640 Dialup : 602-955-4491 @ 8-N-1

The Hidden Reef

Access: Dial-Up
System: TI-99/4a Modified
Software: S&T BBS Software
Location: New York
Content: TI and Geneve file libraries, message bases, door games and e-mail.
Dialup : 718-448-9402 @ 8-N-1

The Keep

Access: HTTP and Telnet
System: Pentium 4 running Windows 2000
Software: Worldgroup BBS Software (up to 256 user connections)
Location: Tigard, Oregon
Content: TI and Geneve file libraries, message bases, door games, multi-user and multiplayer games and e-mail.
Telnet : www.thekeep.net port 23 Web browser to http://www.thekeep.net

The Keep has TI File libraries, Message bases, e-mail, door games, multi-user and multiplayer games. The keep also has a modem line connected for anyone that would like to contact The Hidden Reef BBS from the internet through The Keep.

Simply telnet to www.thekeep.net on port 23, login to The KEEP and then type /**GO DIALOUT** at the main menu, then D1 to dial out to The Hidden Reef. It's that simple.

Vendors

SHIFT838 – Provides used TI equipment as acquired. Check with me often. A lot of the items need rehoming from other TI Users.

Arcade Shopper – Provides old and new TI equipment, upgrades and new runs of PCBs at www.arcadeshopper.com

Repair Centers

Richard Bell
Repairs available on limited basis, please contact Richard at swim4home@verizon.net for wait-time before sending any repairs

Tim
Myarc-related hardware repairs on a limited, as-available basis. Contact Tim at insane_m@hotmail.com for wait times or to request service.