SHIFT838 Newsletter

JULY 17, 2015

VOLUME 1, NO 7

This newsletter is dedicated to the ongoing support for the Texas Instruments TI-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 Harry Wilhelm, designer/creator of some very cool XB tools like Compiler, XB256 and The Missing Link.   These tools enhance the ability to program for a majority of us XB programmers. This is a very detailed explanation of his tools and Harry was kind enough to go into detail of what really is going on behind the scenes with his *COMPILER256*.

Thanks to all that have subscribed!

Interview

Harry Wilhelm

Chris: Can you explain in detail what each of the programs below allows a programmer to accomplish?

Harry: First off, I want to thank Chris Schneider for giving me the opportunity to talk about these programs. As a quick intro to my thinking, my philosophy in writing for XB has always been to write programs that will run on a "standard" TI system - that is TI Extended BASIC, 32K and a disk drive. No special XB is needed, no upgraded graphics card, no memory expansion beyond the usual 32K. I get a kick out of seeing what can be done with equipment that was available in 1981.   Now, taking your questions in reverse order:

THE MISSING LINK

Around 1989 I did some experiments to see whether it might be possible to use the bitmap graphics mode with Extended BASIC, which everyone said was impossible. The idea was to create a utility that used the interrupt routine at >83C4 to automatically detect when an XB program started running and then set the VDP registers for bitmap and clear the screen. The interrupt routine also keeps track of stack pointers and resets them as needed. Another complication is that there are two memory locations that are heavily used by XB (>0370 and >0820). These fall in the middle of either the color or screen image table and cannot be moved, so it was necessary to map the screen image table in an unusual way to avoid having them mess up the image. The first assembly subroutine written for this simply plotted pixels on the screen but it was a real thrill to see an XB program plotting a diagonal line across the screen, returning to the graphics mode when the program broke, then resuming the plotting with CON. After much testing this was shown to be stable and reliable and I decided to expand it with additional assembly subroutines which wound up filling up the entire 8K low memory. In 1990 this was published by Texaments as THE MISSING LINK.

Programs are written in Extended BASIC, which makes them easy to write and easy to understand, and knowledge of assembly language is not required to fully utilize *THE MISSING LINK*. Since none of the normal XB ways to access the screen will work in bitmap mode, these subroutines replace the usual XB commands for accessing the screen.

THE MISSING LINK includes text subroutines that let you input information or display it on the screen. There is automatic word wrap when displaying text and text can even be displayed vertically. You are not restricted to 8x8 or 6x8 characters; different size fonts can be used, all the way down to a 4x6 font which gives 32 rows by 60 columns on the screen. Because it is fully bitmapped, different sized text can be displayed simultaneously on the same screen.

Cartesian graphics allow you to plot points, lines, circles, and boxes. Turtle graphics can be used with none of the ink and color restrictions found in LOGO. Sprite graphics can place up to 32 moving sprites on the screen. There are no limits when combining graphics and text on the screen.

A window can be defined that can be used as a text box. Text is always displayed inside the window; graphics can be displayed either inside or outside the window.

Pictures can be loaded or saved in the standard TI-Artist format and a graphics screen dump is always available (on real iron).

To get an idea of what is possible, RUN “DSK1.TMLDEMO” will run a demo program that shows everything TML is capable of.

Another part of the package is a tutorial called “POTATOHEAD” that shows how to write a simple program similar to “Facemaker” for the TI or “Mr. Potatohead” for those who like to play with their food. If you decide to try out TML, I highly recommend studying Potatohead to get a deeper understanding of how to use TML. If you want to get started without reading the manual, here’s a simple program that draws a series of expanding circles on the screen:
RUN “DSK1.TML” - TML is loaded and activated. The screen is green with a Texas cursor.

```
10 FOR I=1 TO 100 STEP 5::CALL LINK(“CIRCLE”,I,I,I)::NEXT I
20 GOTO 20
```

XB256

XB256 shares its design philosophy with TML. Like TML, it is a collection of assembly language subroutines, but rather than bitmap graphics, with XB256 the focus is on fully utilizing the normal graphics mode. Again, no knowledge of assembly language is required to use *XB256*.

XB256 lets you select from two independent screens. Screen1 is the default screen which is the screen normally used by Extended BASIC. It is accessed using the usual XB graphics statements. Screen2 is completely independent of screen1. It lets you define 256 characters, compared to the 112 normally available to XB. Additionally, you can use up to 28 double sized sprites using the patterns available to Screen1. You can toggle between the two screens as desired while preserving the graphics on each screen. This would be useful for a help screen or menu screen. When using Screen2 there are assembly subroutines that replace CHAR, CHARPAT, COLOR, and CHARSET. Except for these subroutines, all screen access in Screen2 is by the usual Extended BASIC statements such as PRINT, SPRITE, ACCEPT, etc.

There are scrolling routines that allow you to scroll screen characters left, right, up, or down. You can specify a window area for scrolling and leave the rest of the screen unchanged. Other routines let you scroll smoothly one pixel at a time to the left, right, up or down. Yet another lets you do a text crawl like the title scene of “Star Wars” There are miscellaneous subroutines that let you highlight text, set the sprite early clock, print on the screen using all 32 columns, read from or write to the VDP

RAM, write compressed strings to VDP, move sound tables into VDP, play a sound list, freeze and thaw sprite movement, catalog a disk, and more.

A utility (COMPRESS) is included that lets you save selected areas of VDP memory into compressed strings that can then be merged into your program. You can save character definitions, sound tables, screen images, etc. in a compact form that can be loaded much more rapidly.

Another useful feature is the ability to play sound lists. Sound lists are a very compact way to save music and sound effects in a compressed form that can be loaded directly into VDP memory and played automatically while your program does other things. The sound list player in XB256 can play two sound lists simultaneously so you could have background music continuously playing and add sound effects such as explosions on top of the music. No need to learn assembly to create the sound list. Starting with an XB program that makes music with CALL SOUND statements, there are two different utilities, SLCOMPILER and SLCONVERT that can be used to convert the CALL SOUND statements into a sound list that can then be merged into an XB program.

COMPILER 256

Although Forth enthusiasts might take issue, Extended BASIC language is arguably the most versatile of the languages available for the TI99/4A. Programs are easy to write, relatively understandable, and simple to modify and edit, with lots of error checking to facilitate program development. The main drawback is that the double interpreted nature of Extended BASIC makes it extremely slow.

The intent in writing my Extended BASIC compiler was to make it possible to take full advantage of the simple program development offered by XB, then make an end run around the speed limitations. The goal was to implement Extended BASIC as fully as possible within the time limits of the programmer and the memory limits of the machine. There are limitations and you will probably need to adjust your programming style a bit, but in general, all the major features of XB are included in the compiler. This means that you can concentrate on writing the XB code and testing it in the XB environment. After the program has been perfected in Extended BASIC it can then be compiled into an equivalent code that functions at a speed approaching that of assembly language. The average Extended BASIC program will run about 20 times faster after being compiled, and certain operations will run up to **70** times faster.

There are several methods by which the compiler achieves this speed increase. First, Extended BASIC must perform a lengthy prescan operation before a program can even start. This is done in advance by the compiler and becomes part of the compiled code. Second, an XB program is interpreted twice by the computer; once by the Extended BASIC interpreter, and a second time by the GPL interpreter. The compiler generates "threaded code" which needs its own interpreter (called the runtime routines), but at least only one interpreter is involved, and it's a fast one! Third, integer arithmetic is used throughout instead of floating point arithmetic. This alone makes the code run at least 5 times faster, admittedly without the versatility of 13 digit floating point accuracy. Fourth, to increase the speed even more, virtually no error trapping is done. Any error reports that are given are not very helpful anyway because you won't know the line number where the error happened. Therefore it is imperative that the Extended BASIC program be thoroughly debugged before you attempt to compile it!

The compiler has been expanded to include all the *XB256* assembly language extensions.

You can test your program in the XB or *XB256* environment, then use the compiler to get a huge performance increase. The compiler has been tested with a genuine TI-99/4a, with Classic 99, and Win994a and is compatible with all of them. A program can be saved as an XB program or optionally as an EA5 program.

XB256 and *COMPILER256* are at their best when used together, and the combination is the XB Program Developer's Package. Although they are designed to complement each other, both these are standalone utilities. Programs developed using XB256 do not have to be compiled. If the execution time is adequate they will run fine in XB, and you would have access to floating point math, disk access, etc. Similarly, XB programs do not have to use *XB256*; they can still be compiled for the increased speed.

Now I have to bring up an unpleasant subject: If you decide to use any of these programs, keep the manual close at hand and refer to it often. If you do that you will make much faster progress!

Chris: There have been quite a few questions around the compiler and how it actually functions. Can you provide (at a high level) basically what it is doing when generating the new code?

Harry: Ideally a compiler would take an XB program, convert it directly into a ready to run assembly language program, and save it to disk. But this approach would be very difficult to write and debug, at least for me. To try to preserve my sanity, my approach was to break the compiler into two parts. The first is the actual compiler that does the conversion. This is a hybrid XB/assembly program that converts a merge format XB program into what is called "threaded object code." This code closely follows the original XB program. As an example of how it works, the short program below prints "Hello World!!" on the screen using the TI BASIC method; then the compiled version follows:

```
10 A$="Hello World!!"
15 CALL CLEAR
17 FOR R=1 TO 19
20 FOR I=1 TO LEN(A$):: CALL HCHAR(R,R+I,ASC(SEG$(A$,I,1)))::NEXT I
30 NEXT R
40 GOTO 40
```

Here is what results when the compiler has converted the program. My comments and the XB program are in italics. Notice how the compiler breaks the complicated CALL HCHAR into simple component parts.

```
     DEF RUN,CON
RUNEA  B @RUNEA5                      Only used by an EA5 program
FRSTLN                                The first line of the program.
L10                                   10 A$="Hello World!!"
     DATA LET,SV1,SC1                 A$ (SC1) is stored in SV1
L15                                   15 CALL CLEAR
     DATA CLEAR
L17                                   17 FOR R=1 TO 19
FOR1
     DATA FOR,NV1,NC1,NC2,ONE,0,0     set up the for/next loop. R is now
                                      NV1
L20                                   20 FOR I=1 TO LEN(A$):: CALL
                                      HCHAR
                                      (R,R+I,ASC(SEG$(A$,I,1)))::NEXT I

     DATA LEN,SV1,NT1                 First length of A$ (SV1), put in NT1
                                      for use in loop
FOR2
     DATA FOR,NV2,NC1,NT1,ONE,0,0     Set up the For/Next loop; I is now
                                      NV2
                ( Here the compiler breaks the line into simpler steps : )

     DATA SEGS,SV1,NV2,NC1,ST1        Take SEG$ of A$, store in ST1
     DATA ASC,ST1,NT1                 Store ASC of ST1 in NT1
     DATA ADD,NV1,NV2,NT2             Add NV1 to NV2, store in NT2
     DATA HCHAR,NV1,NT2,NT1           Now simplified enough to do HCHAR
     DATA NEXT,FOR2+2
L30                                   30 NEXT R
```

```
       DATA NEXT,FOR1+2
L40                                              40 GOTO 40
       DATA GOTO,L40
LASTLN DATA STOP
OPTBAS DATA 0
NC0
ZERO   DATA 0
ONE    DATA 1
PI     DATA 3
RND    DATA 0
NC1    DATA 1
NC2    DATA 19
NV0
NV1    DATA 0 R
NV2    DATA 0 I
NT1    DATA 0
NT2    DATA 0
SC0
SC1    DATA SC1+2                                Convert "Hello World!!" to bytes; this
                                                 way any ASCII can be used in a string
       BYTE 13,72,101,108,108,111,32,87,111,114
       BYTE 108,100,33,33
       EVEN
SV0
SV1    DATA 0 A$
ST1    DATA 0
SA0
NA0
FRSTDT
LASTDT
       EVEN
       COPY "DSK1.RUNTIME1"                      copy the runtime routines
       COPY "DSK1.RUNTIME2"
       COPY "DSK1.RUNTIME3"
       COPY "DSK1.RUNTIME4"
       END
```

You can see that the compiled program closely follows the original XB program, which makes debugging easier. Now it's time to do something with the code that was created. There is a main loop that is the key to running the code:

```
STAR8  LI R13,FRSTLN          R13 points to the place in compiled code, we start
from 1st line
*********************         then later on:
RTN    LIMI 2                 the main loop, all subs come back to here
       LIMI 0
       MOV *R13+,R12          move the address of the code into R12 and add 2 to
                              R13
       B *R12                 branch to the code pointed to by R12  i.e. GOTO,
                              SPRITE, etc.
                              return is done with  B @RTN
```

The subroutines in the runtime routines automatically update R13 so when they return they will point to the next instruction. Here is the LEN subroutine which is part of RUNTIME2:

```
LEN    BL @GET2               get the 2 words following LEN, put in R5 and R6, add
                               4 to R13
       MOV *R5,R5             now R5 points to the string (1st byte is length byte)
LEN1   MOVB *R5,R4            move length byte to MSB of R4
       SRL R4,8               shift length byte to LSB
       MOV R4,*R6             return the length to the address pointed to by R6
       B @RTN                 all done, go back. GET2 updated R13 to point to next
                               instruction.
```

So basically the runtime routines fetch the address of each compiled instruction (SEG$, LEN, PRINT, CALL HCHAR and all the rest). The instruction is branched to. If any parameters are passed then R13 must be updated so that when B @RTN is performed it returns with R13 pointed to the next instruction. This is a brief overview of how the compiler works; hopefully simple enough to follow. As you might expect the runtime routines can get a lot more complex than the simple example above, but all are branched to by the main loop and return to it.

Chris: I know there are speed benefits with the compiler. For a typical program what type of percentage as far as faster execution speed would one see?

Harry: As mentioned above, the speed increase is usually at least 20 times faster and sometimes 50 times faster or even more. Also, programs start up immediately with no wait for the prescan to happen. My real TI99 has 32K on the 16 bit bus, so compiled programs will run even faster by 20 to 25%. When writing games you usually have to add delay loops to slow things down enough to make the game playable.

Chris: I know there are limitations with File IO and the compiler is there any plan on addressing that and if so when?

Harry: The compiler has been updated and can now handle disk access. Only Display, Variable is supported, but you can use any length from 1 to 254. Other file types could be supported; some of the framework is there. It would just be a matter of expanding the runtime routines (RUNTIME7 to be exact). Remember that the runtime routines become part of the compiled program. Every byte added to them means one less byte available to your program. I am trying to strike a balance between completeness and having enough room for your program.

Chris: Are there any compatibility issues for any of the above programs in question #1, and if so what?

Harry: The only compatibility issue I am aware of is that The Missing Link cannot use any disk access if you have the CF7 expansion system. If I had a CF7 available this could probably be fixed, but my efforts to do it from a distance have been unsuccessful.

Chris: Are there any restrictions that a user needs to be aware of before they use any of these programs?

Harry: The biggest restrictions are in the compiler. Not all of XB is implemented and if you try to compile an XB program with unsupported instructions you will have trouble. You are restricted to integer arithmetic, cannot use named subprograms, nested arrays, trig functions, speech, display using, DEF and a few other things. IF THEN ELSE works line numbers (like BASIC), not the more complex form possible in XB. These are all detailed in the manual. It is best to develop a program to be compiled keeping these restrictions in mind, rather than converting an existing program. If you want to compile an already written XB program you will likely have to go through it and see if unsupported instructions are used. Some programs are easier to modify than others - I did successfully modify and compile the XB program “Portal” which now has a nice snappy response. Existing TI BASIC programs are more likely to compile without modification than XB programs.

The entire compiled program is loaded into high memory as well as the runtime routines and string variables, which normally are contained in VDP ram. Compensating for this is that the compiled code is usually around 2/3 the size of the original XB program. So if you have a very large XB program it should still be possible to compile it and have it fit in memory.

The Missing Link uses most of the VDP ram for the graphics. The ram available for programming is the same, but the stack space in the VDP ram is very limited.

Strings are kept in the stack and if you have a lot of them you can easily run out of stack space. There is a section in the manual that gives information on how to save stack space.

Chris: Where can these programs be obtained?

Harry: These are available in two places:

Atariage TI99 forum
http://atariage.com/forums/forum/119-ti-994a-development/

You will find them in TI-99/4A development resources at the top of the page.

http://www.99er.net
Go to Download Database, then go to Utilities and you will find them

Hardware Upgrade

TI-99/4A Reset Switch Installation

This subject has been brought up multiple times so I decided to look into this for this edition of the newsletter. Users of the TI-99/4A like myself have often wondered why Texas Instruments did not design this computer with a reset switch. As we have all probably experienced a hard lock-up of the system and the only solution is to flip the power switch. Of course this can cause a lot of wear and tear on the switch. I myself have experienced a power switch failure on the unit and have had to replace it.

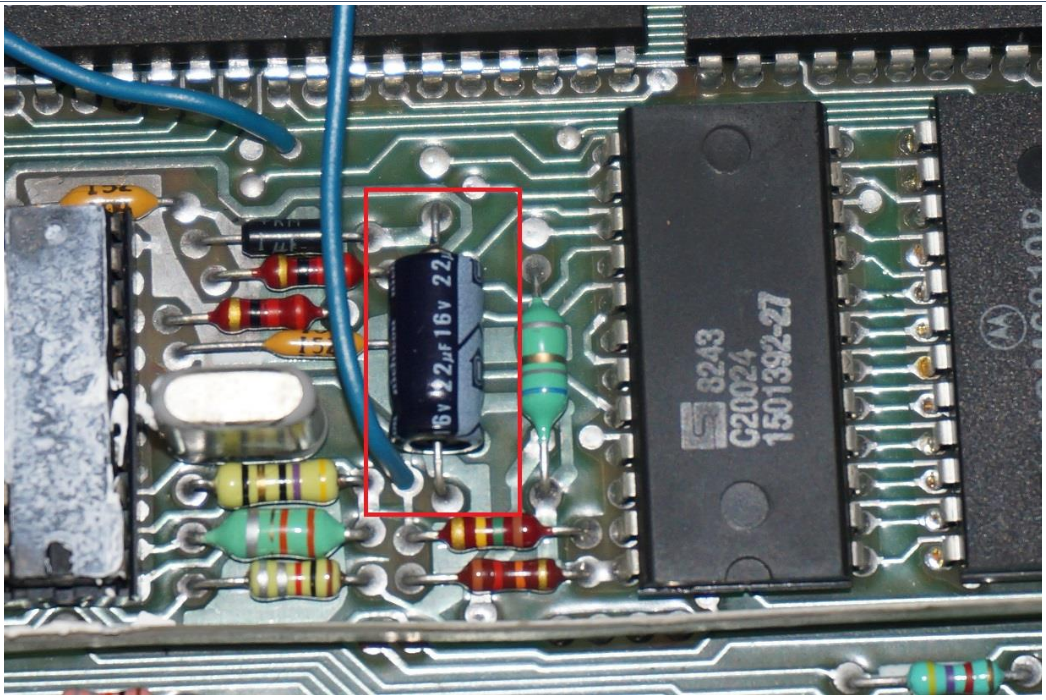
After much questioning and researching of the modification, I wanted to publish what I thought was a good solution for all level of users. I have chosen this solution as this does not require any soldering to the CPU or cutting of traces on a board and is an easy upgrade for anyone, even the beginner.

Parts List:

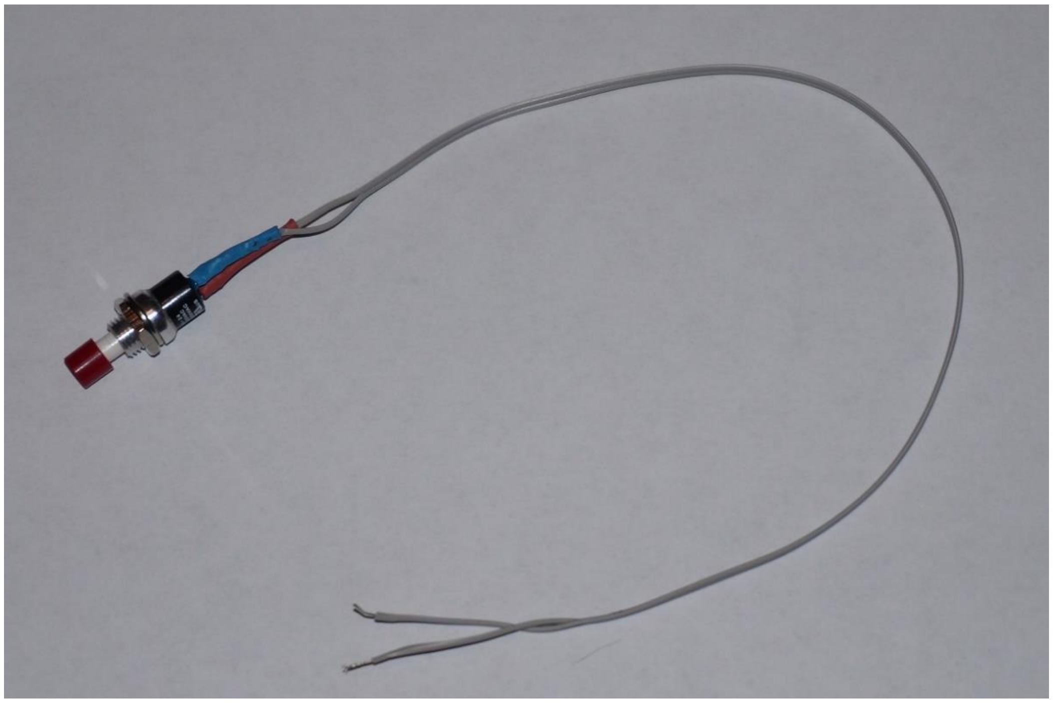
1 x Momentary Pushbutton Switch (Normally Open)
2 x 12 Inches of wire (Ribbon cable works great)
Some small shrink-tube or electrical tape

Installation Steps:

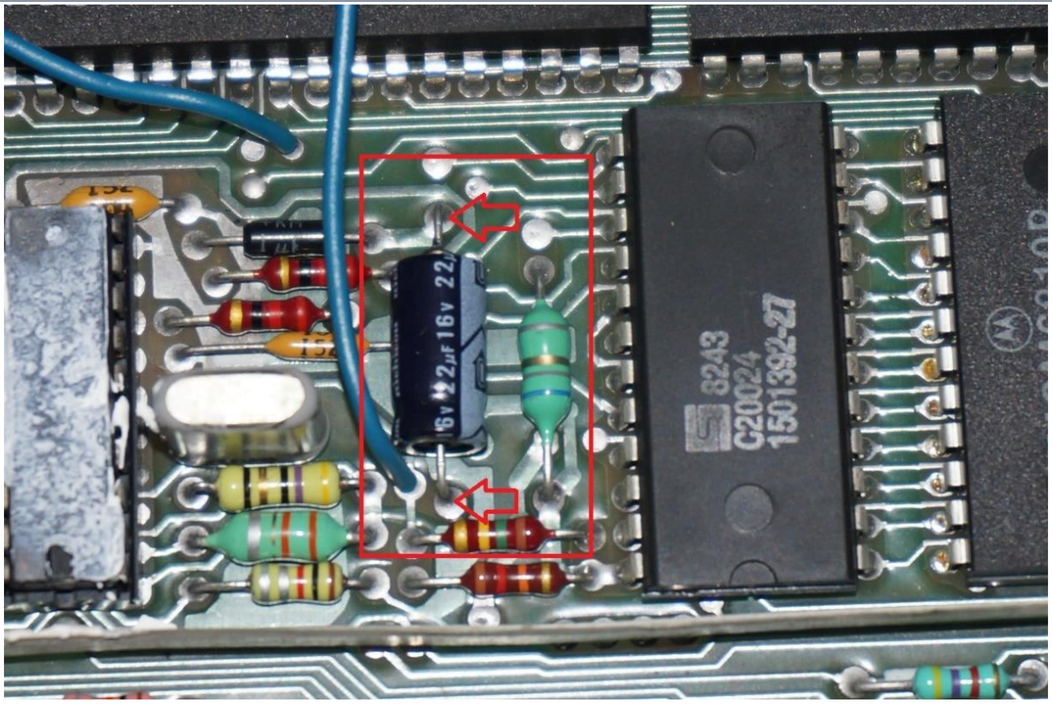
Step 1: Disassemble Console	Good disassembly steps can be found on http://www.mainbyte.com
Step 2: Locate Capacitor C606 –	This capacitor is located on the right of the grey crystal under the TMS9900 processor



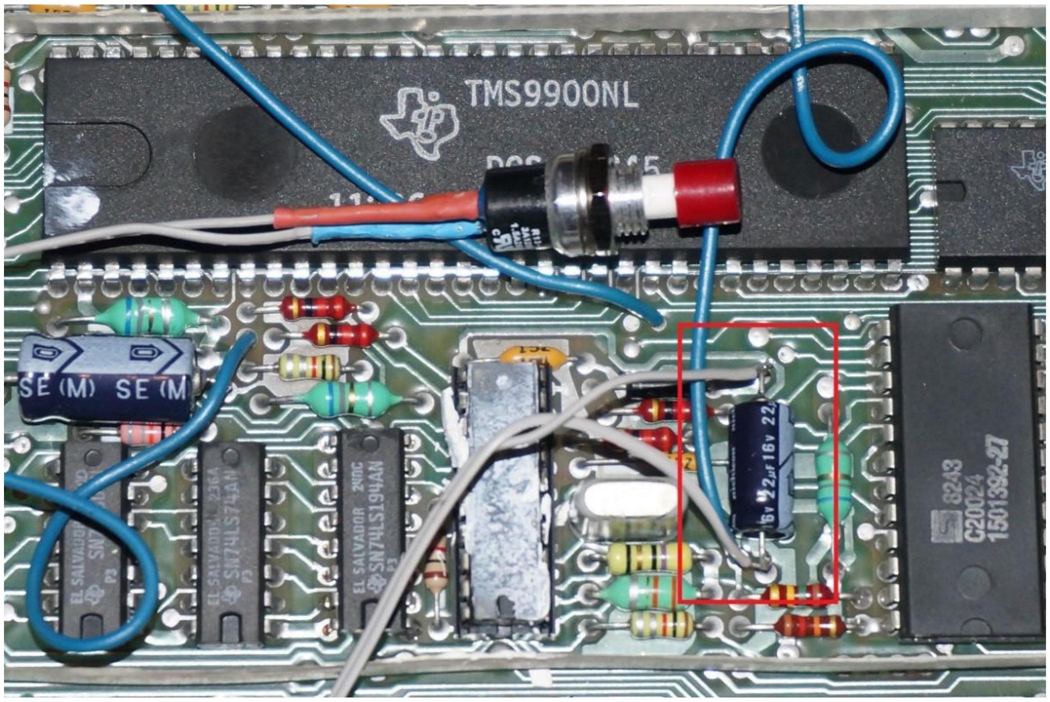
Step 3: Solder Cables to momentary Switch and use 2 pieces of shrink-wrap tubing to cover terminals.



Step 4:Solder wires from momentary switch to capacitor C606 (one on each side of the capacitor)



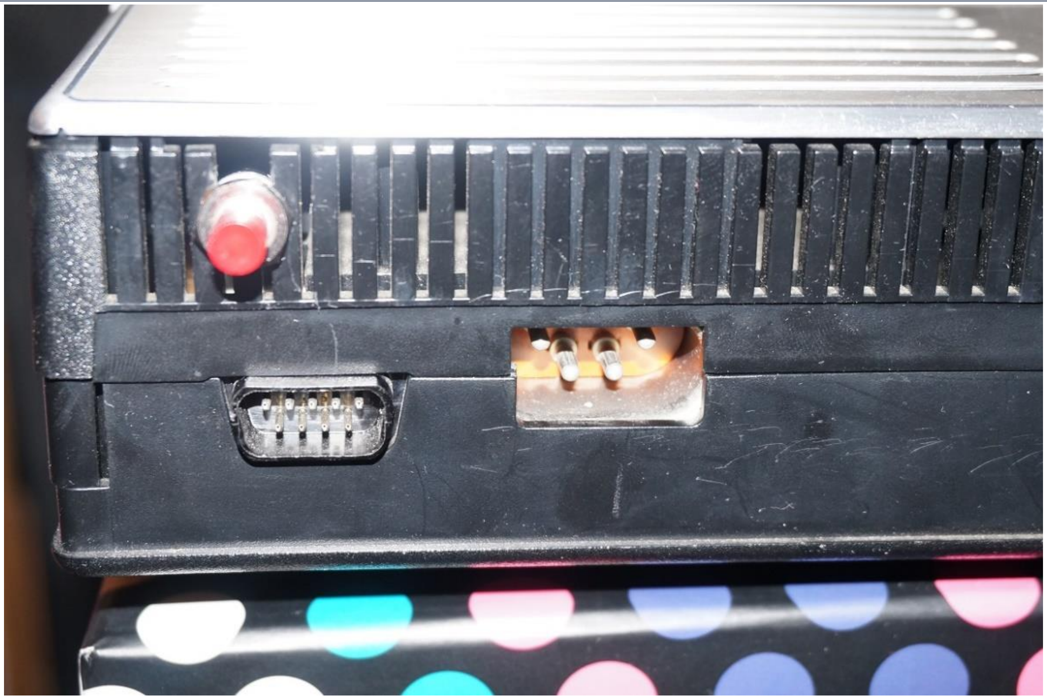
Once soldered it should look as the below picture does.



Step 5: Clip a center vent hole on the top back right of the console to mount the switch. (May need to use a round file as well to get a good fit).

Step 6: Mount momentary switch

Step 7: Reassemble the console



You now have a reset switch. You can mount the switch anywhere and it can be any type of momentary push-button switch as long as it is 'Normally Open'. If you choose to mount in other locations just make sure to use the appropriate amount of wire and the switch does not interfere with the console in any way.

Pressing the switch will reset the console without having to engage the power switch.

Software

SoftWorx BBS V1.0 – This was the first BBS that I ran and was written by Mark Shields that ran the USS Enterprise here in Houston, TX. I am currently trying to revive his latest version of ***SOFTWORX***.

This BBS is ran off the original Zyolog Machine Language coded by Bryan Wilcutt. The XB code was written from the ground up by Mark Shields. This BBS supported messages bases, text files, and XMODEM and ASCII file transfers.

Mark's coding ability was very good and only grew more with his progression of understanding the TI-99/4A.

I thought that Mark used a very unique method for detecting a carrier back in the early 80's. He used the joystick port wired to pin #8 of the modem (DCD) to pin #9 of the Joystick port. This method would not return a value if a carrier was detected but would return a value > 0 if a carrier was not detected. For a more in depth break down see the '**Coding**' section below.

Remember not to plug the joystick cable in until it tells you to.

This BBS has to be ran on a real TI system. I have verified it will not work via any emulator nor will it work with the NanoPEBs. It can be ran from floppy drives, RAMDisk and hard drives.

The software can be downloaded on the FTP site under my directory.

ftp://ftp.whtech.com/Users/Chris_Schneider/BBS/SOFTWORX_V1.0

I have also put together a very small document to explain the **CALL LOADS** as well as the **CALL LINKS** that are used in the code. I had to do some testing to figure out exactly what they were doing since I have not messed with this program since the mid 80's.

If you want to play with it then by all means have fun. If you have any questions please feel free to contact me and I will do my best to answer them.

You can also read more about other vintage BBS software for the TI-99/4A at the below link:

https://en.wikipedia.org/wiki/BBS_software_for_the_Texas_Instruments_TI-99/4A

Coding

The below code example shows a quick program to detect a carrier. I have verified this works on both a real modem and UDS device with real TI hardware.

A cable will need to be made that connects PIN #9 of the joystick port to pin #8 on the back of the modem/UDS device. This is the Data Carrier Detect pin. When a carrier is present the value will be a '0'. When the programs does not detect a carrier it will return a value greater than zero.

Lines 100 to 170 really just make the program pretty as it changes the '-' character to be solid and makes it look like one continuous line on each row.

Note: No modem connection should be active when executing the program.

```
100 ! MODEM CARRIER DETECT
105 ! ORIGINAL CONCEPT BY MARK SHIELDS
110 ! (C)OPYRIGHT 2015
120 ! CHRIS SCHNEIDER
130 !     SHIFT838
140 CALL CHAR(45,"00000000FF")
150 CALL CLEAR
```

NO CARRIER should be present at this time.

```
160 PRINT "Plug in joystick DCD cable!"
170 CALL KEY(0,K,S):: IF K=49 OR K=80 THEN 180 ELSE 170
180 CALL CLEAR
190 DISPLAY AT(6,1):"----------------------------"
200 DISPLAY AT(8,1):"----------------------------"
```

The below code is where it all starts to check for carrier continuously. You can now connect to your modem from another machine and see the status change from '**NO CARRIER DETECTED**' to '**CARRIER DETECTED**'. When you drop carrier it will go back to '**NO CARRIER DETECTED**'

```
210 CALL JOYST(1,X,Y)
220 IF X>0 THEN 230 ELSE 250
230 DISPLAY AT(7,5):"                   "
240 DISPLAY AT(7,5):"NO CARRIER DETECTED" :: GOTO 210
250 DISPLAY AT(7,5):"                   "
260 DISPLAY AT(7,5):" CARRIER  DETECTED "
270 GOTO 210
```

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

A CHAT application has been added to the site for active live chat.

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.