TI99IUC.IT NEWSLETTER

Traduzione della newsletter originale SHIFT838 di Chris Schneider tradotta da A. Cevolini

14 Giugno 2015 **VOLUME 1, N°6**

Questa newsletter italiana nasce da un accordo avuto con Chris Schneider che ha acconsentito alla traduzione italiana della sua *Newsletter Shift838*. Vuole essere una nuova risorsa per i computer **TI-99/4A** e **GENEVE 9640**, in modo da divulgare e mantenere aggiornati tutti gli utenti che ancora oggi si interessano a questi computer.

In questa newsletter si parlerà con ***Matthew Hagerty***, progettista della scheda F18A – la nuova scheda video per il TI-99/4A ed altri sistemi.

Altri argomenti trattati:

- **L'Avventura "*The Stafford Predicament*"** (by Chris Schneider)**;**
- **Codifica dei file I/O con il TurboForth** (by Mark Wills)**;**

Buona Lettura !

Ciro Barile

Intervista a:

Matthew Hagerty

D: Quando nacque l'idea della scheda F18A e che cosa te l'ha fatta pensare?

R: Nel 2005 o giù di lì ho partecipai al TI Faire a Chicago per la prima volta. Nella sessione di "retrocomputer scene" appena dopo che Jeri Ellsworth diede dimostrazione della scheda C-One (C64 su singolo chip – SoC ndt), andammo tutti in pausa e fu in quel momento che arrivarono domande sulla FPGA (circuito integrato FIELD-Programmable Gate Array). Ci feci un pensierino, ma poi mi resi conto che qualcun altro avrebbe potuto realizzare un SoC per tutti gli home computer classici molto più rapidamente di me, seguendo l'esempio di Jeri. Poi un'idea mi tornò in testa e nel 2008 comprai una devboard FPGA, che però subito dopo riposi su una mensola dove è rimase fino a quasi il 2011.

Capitò che, probabilmente nel 2011, qualcuno sul forum palesò l'idea di ottenere una migliore uscita video dal TI99/4A, o sostituendo il 9918A con un 9928 per ottenere un video component, o per la conversione analogica esterna del segnale composito del 9918A. Tali progetti non hanno però mai prodotto un sufficiente miglioramento per cui valesse la pena realizzarlo oppure richiedevano così tante modifiche (saldatura, taglio di tracce, ecc) al TI99/4A, tali da impedire a molti utenti di poter ottenere un pur minimo aggiornamento. Così iniziati a pensare come il video avrebbe potuto raggiungere una qualità migliore, cosa che comunque è insita nel 9918A in termini di raw data prima di fornire l'uscita video.
Sono sempre stato affascinato dalla grafica del computer e ho sempre desiderato sapere come i sistemi di videogioco del tipo coin-op potessero generare video dal momento che non utilizzano nemmeno chip VDP come il 9918A. Mi decisi di voler provare sostituire il 9918A usando un FPGA e questo mi ha permesso di imparare tutto su questi circuiti, ed è proprio con questa motivazione che ho imparato l'HDL (linguaggio di descrizione hardware), l'architettura digitale, l'FPGA per interfacciare l'esterno del circuito, il circuito di progettazione, il layout e la produzione.

Sinceramente ho sempre pensato che sarebbe stato bello poter fare una semplice sostituzione del tipo "drop-in" del 9918A tale da permettere agli utenti di poter facilmente innestarlo come "plug-in" con il minimo di sforzo (solamente smontando la console), ma purtroppo avevo altro da fare e poi dovevo comunque continuare con il mio lavoro.

D: Perché la F18A è compatibile con tanti sistemi?

R: La F18A è compatibile con tanti sistemi perché gli originali 9918A/9928/9929 VDP sono chip video generali e sono usati in molti sistemi. La F18A è stata progettata con la stessa configurazione fisica dei pin, stesse specifiche elettriche e stessa interfaccia del timing dei VDP originali. Il circuito F18A è un po' più largo rispetto allo zoccolo DIP a 40 pin originale (di ciò non sono felice), ma a parte questo, può sostituire direttamente le VDP originali.

D: Quali elementi del 9918 sono stati i più difficili da implementare?

R: L'elemento ceh mi ha dato più del filo da torcere riguardo la VDP originale è proprio quello che non ho implementato: l'uscita video composito a colori.
Anche se l'obiettivo principale della F18A era quello di produrre un segnale pulito ad alta risoluzione video tipo il VGA per gestire i monitor moderni, mi saprebbe piaciuto anche provare a lasciare l'uscita video composita originale. Inutile dire che, mentre l'uscita VGA è facile, il video composito a colori NTSC/PAL non lo è per niente. Ho speso un sacco di tempo, denaro, risorse dell'FPGA e spazio fisico sulla scheda cercando di ottenere un'uscita composita aggiunta.

Internamente gli elementi più difficili erano quelli che sono vagamente documentati o non documentati affatto, come per esempio il 5° sprite nel registro di stato, l'attuale frame interrupt trigger (può variare entro poche linee di scansione, ma non è noto esattamente perché), ecc. Questo tipo di funzionalità è nota solo attraverso la caratterizzazione dell'originale VDP e approfondire quel tipo di informazioni prende un sacco di tempo, per la ricerca e per la sperimentazione.

Inoltre, mentre stavo ancora imparando la progettazione digitale di circuito, del VHDL e del FPGA in generale, ho scoperto come implementare determinate funzionalità fosse molto difficile. Mentre cercavo di aggiungere sprite e circuiti, sono arrivato ad un punto in cui mi ero piantato. E' stato a quel punto che presi un po' di tempo, acquistai altri libri e smisi di pensare in termini di programmazione. Dal momento che arrivavo da un background da programmatore software e che la sintassi dell'HDL 'sembra essere simile' alla programmazione, mi è capitato di commettere vari errori operando con la mentalità del softwarista. La transizione a una mentalità da hardwarista è stata per me vincente e mi ha permesso di completare il progetto e migliorare i circuiti.

D: Ci sono ovviamente diverse restrizioni/limitazioni sul vecchio hardware che potevano essere facilmente eliminate nella nuova architettura FPGA ... Che tipo di cose hai avuto bisogno di conservare per mantenere la compatibilità pregressa con il software esistente?

R: Gli sprites ne sono probabilmente il miglior esempio. Essere in grado di visualizzare tutti i 32 sprites su una linea in una sola volta è possibile con la F18A e ho inizialmente pensato "Non c'è più nessun sfarfallio! Chi non lo vorrebbe??" ☺ Tuttavia, mentre la maggior parte del software ha patito la limitazione dei quattro sprites per linea, ho scoperto a mie spese che alcuni software si basano su tale limitazione. In alcuni software il fatto, che solo quattro sprites per linea possono essere visualizzati, è usato per mascherare altri sprites, individuare la linea di scansione orizzontale, ecc. Il numero predefinito di sprites su una linea (4 al posto di 32) mi ha indotto ad aggiungere dei ponticelli di settaggio fisici sul circuito F18A. Essere in grado di visualizzare tutti i 32 sprites, ma di mantenere l'originale funzionalità/compatibilità è una cosa su cui ho lavorato per tutto il tempo fino all'ultimo firmware V1.6.

Implementare e conservare solo le funzionalità originali sarebbe stato molto più facile. Non appena si inizia ad aggiungere nuove funzionalità e ad ampliare il progetto, occorre stare molto attenti in quanto apparentemente cambiamenti dissociati possono influenzare il software esistente (e anche l'hardware!) in modi di cui non ci si rende conto.

D: Dopo il rilascio iniziale e il successivo feedback da parte degli utenti su più sistemi, quali sono stati i suggerimenti che hai ricevuto sui bug o quali le richieste sulla funzionalità?

R: I bugs dapprima. Credo che il primo problema segnalato sia stato il gioco Penguin in XB, dove sono state rilevate collisioni non corrette tra gli sprite. Questa cosa che un gioco XB, fra i tanti, presentasse un problema mi stupì. Come si è visto, la F18A rileva le collisioni tra sprite off-screen (che la scheda tecnica del 9918A ti porta a credere che possa accadere), ma la vera 9918A rileva solo le collisioni sui pixel dello sprite sul display attivo, cioè nell'area dei 256x192 pixel. Utilizzando CALL COINC (ALL) in XB compare il bug perché gli sprite off-screen nella F18A segnalano il bit di collisione nel registro di stato della VDP.

Ci sono state poche altre segnalazioni di bug che portarono alle versioni V1.4 e V1.5 del firmware. Gli aggiornamenti del firmware sono un problema perché non tutti installano l'aggiornamento e in questo momento l'unico aggiornamento disponibile del sistema è per il TI-99/4A. Originariamente avevo programmato di

includere sulla scheda un interfaccia USB per gli aggiornamenti, ma ancora una volta tempo, denaro e spazio sulla scheda mi hanno costretto a tagliare via dalle mie idee questa funzione.

La prima richiesta ricevuta dagli utilizzatori fu sulle caratteristiche che avrei potuto implementare, per esempio una richiesta di più VRAM, di un'uscita HDMI, di essere più economica o di ridisegnare la scheda, è stata di **Rasmus** *(uno dei maggiori programmatori di giochi che sfruttano la F18A sul TI-99/4A ndt)*.

D: Quanto ha giocato il feedback degli utenti per il nuovo aggiornamento del firmware?

R: Il feedback e le richieste sono state una motivazione enorme che hanno fatto uscire ol firmware V1.6, così come i cambiamenti aggiuntivi per supportare i poco conosciuti 9118A/9128/9129 VDP che furono di breve durata e furono in seguito sostituiti dai 9918A/9928/9929 con cui siamo più familiari.

Le modifiche sulla V1.6 iniziarono quando Rasmus provò alcune delle "caratteristiche" originali e ci rendemmo conto che erano praticamente inutili. Questo è un problema quando si sviluppano caratteristiche fuori dal contesto e non si dispone di nessun riferimento in merito al fatto che una caratteristica potrà essere utile o meno. Iniziai a fare le modifiche in base al feedback e più ne ho inserite, più Rasmus ne ha chieste "ehi, questo sembra facile, si può aggiungere questo?" e "sarebbe bello se potessimo fare ..." ☺
I cambiamenti sulla V1.6 iniziarono nel mese di aprile 2014, poi mi presi una pausa perché in quel periodo cambiai lavoro, ma complessivamente in 6 a 8 mesi, riuscii a riscrivere totalmente un sottosistema con funzioni nuove e utili.

Se non fosse stato per i commenti ed i suggerimenti, la V1.6 avrebbe probabilmente corretto solo i pochi ultimi bug. Invece il firmware V1.6 include massicce modifiche e tonnellate di nuove caratteristiche.

D: Potresti descrivere brevemente alcune delle caratteristiche più interessanti del nuovo aggiornamento del firmware?

R: Per me una delle caratteristiche più entusiasmanti e interessanti disponibili nel nuovo firmware V1.6 è la sezione al secondo livello che consente di inserire un secondo tile layer sopra al primo. Entrambi i tile layer sono completamente indipendenti e possono essere a scorrimento separato dei pixel (orizzontale e verticale) ed entrambi hanno la loro tabella dei nomi e la loro tabella dei colori. Rasmus ha fatto un fantastico set di programmi dimostrativi per il firmware V1.6 e

la demo al 2-plane demo mette in mostra splendidamente la nuova funzionalità.

Un'altra caratteristica interessante è la possibilità di alternare i tile layer aggiungendo le modalità di testo (T40 e T80). Ciò consente di assegnare il colore al primo piano e allo sfondo in modo indipendente. Questa è una grande caratteristica per gli emulatori di terminali in quanto possono ora mostrare veri colori ANSI sui sistemi BBS. Inoltre permette anche ad un editor avanzato di testo di visualizzare un testo evidenziato, una codifica a colori e altre cose simili. Alcune altre chicche degne di nota sono un uso migliore della VRAM per migliorare il colore (più colori per pixel), il supporto di scorrimento per le modalità di testo, un timer di altissima precisione, un GPU HSYNC e un trigger VSYNC e altro ancora.

D: Come prevedi, in seguito, di progredire nell'applicazione e nello sviluppo grazie alle nuove funzionalità del nostro vecchio sistema (dovute alla scheda F18A)?

R: Io non sono sicuro su come la F18A influenzerà lo sviluppo nel futuro. Se la storia è un indicatore del futuro, purtroppo sembra che sia piuttosto desolante. Non sto certo riducendo l'importanza del fatto che grandi programmi sono stati sviluppati utilizzando alcune caratteristiche della F18A, ma la F18A è stata rilasciata a partire dall'estate del 2012 e non c'è ancora un nuovo software che sia stato scritto unicamente per la F18A.

Tutte le applicazioni che supportano le funzionalità della F18A supportano anche il 9918A VDP originale e vi è attualmente un grande interesse per spingere la VDP originale e fare cose che avremmo pensato, in precedenza, impossibile. Penso che sia molto bello vedere queste cose sul VDP originale e se avessimo visto che questi esempi potevano farsi prima che io iniziassi a fare la F18A, probabilmente non avrei implementato lo scrolling dei pixel o alcune altre caratteristiche.

Niente di tutto questo mi induce a lamentarmi e capisco che le persone che scrivono software vogliono renderlo disponibile al più ampio pubblico, cioè alla gente che attualmente è senza una F18A. Tuttavia, questo fatto suggerisce anche che lo sviluppo del software non cambierà molto in futuro.

D: Quali sono alcune delle innovazioni che prevedi per la F18A in futuro per il TI-99/4A

R: Non sono sicuro di cosa si intenda per avanzamenti, ma la F18A è un pezzo di hardware ormai consolidato e i progressi dovrebbero riguardare per la maggior parte solo correzioni di bug. Per ottenere maggiori funzionalità dal dispositivo esistente occorrerebbe la riprogettazione della gran parte della circuiteria.

D: Hai pensato di rilasciare forse un kit di sviluppo software per la F18A che permetterebbe agli utenti di utilizzare le funzionalità F18A per ambienti di programmazione?

R: Assolutamente sì, sto pensando a esempi di codifica, documentazione, ecc. ma per tutto ciò mi servirebbe il tempo di una seconda vita. Ho una famiglia e un lavoro giornaliero, così devo sempre mettere gli hobby in secondo piano.

D: Per esempio:

- **la programmazione in Extended BASIC di un comando per caricare un programma nella High Memory (da >A000 a >FFFF) mediante delle CALL LINK**

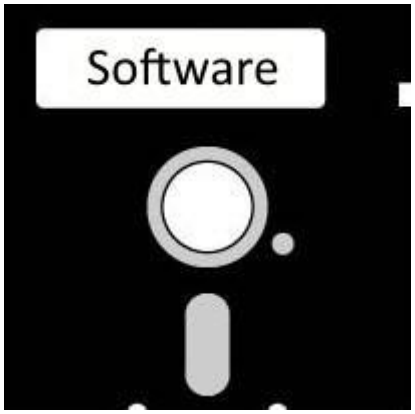
R: Non sono sicuro di riuscire a tanto con l'XB. Il problema più grande con XB è evitare di entrare nell'XB. Devi operare molto "vicino" al sistema e così è facile fare un passo falso e danneggiare l'ambiente. Molte delle grandi caratteristiche dell'F18A richiedono il controllo e l'uso delle VRAM che potrebbero causare problemi all'XB.

- **programmi in assembly sul DIS/FIX 80 dell'Editor/Assembler (opzione #3 [opzione 4 nella suite dell'XB v2.7]) e sul PROGRAM image file (opzione #5 [opzione 6 nella suite dell'XB v2.7])**

R: Certo, l'Assembly è attualmente il migliore ambiente per sfruttare la F18A. Non sono sicuro di quello che potrei fornire per quanto concerne gli strumenti in futuro, ma qualche esempio e la documentazione sono sulla mia lista di cose da fare.

- **TurboForth ?**

R: Non sono sicuro di quello che posso fare per i programmatori TF (TurboForth) se non quello di supportare Willsy nell'implementare la F18A nel TF.
Attualmente nel TF già supporta T80 sul F18A, ma non so cos'altro sia previsto.
Se il TF consente l'accesso diretto ai registri VDP e non fanno ipotesi su come viene usata la VRAM, allora i programmatori di TF hanno già tutto ciò di cui necessitano per trarre il massimo vantaggio dalla F18A.



The Stafford Predicament *(L'imbroglio di Stafford)*

By Chris Schneider (Shift838)

Questo è un gioco di tipo Adventure testuale da usare con il rispettivo modulo "Adventure". Utilizza il 97% della memoria disponibile ed è molto esteso. Di gran lunga la più ampia avventura programmata da Chris che ha investito più di 100 ore di codifica. In questa gioco c'è un Easter-Egg che devi cercare di scoprire ma prima devi scoprirlo (senza utilizzare l'editor per favore!).

L'avventura è scaricabile dai seguenti link:

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

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

Coding

Volevo soffermarmi su alcuni accessi base I/O usando il TurboForth per continuare quanto riportato nelle newsletter degli ultimi mesi.

Sono stato in contatto con Mark Wills per avere un esempio di alcuni file di I/O. Mark ha messo qui sotto un po' di materiale da pubblicare in questa newsletter mensile. Grazie Mark, questo aiuterà molti di noi ad avere una migliore comprensione del file di I/O del TurboForth.

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

Aprire file di testo in TurboForth

Recentemente mi è stato chiesto come è fatto un semplice file di I/O in TurboForth così ecco un articolo sbrigativo per mostrare come leggere un file di testo (DV80) dal disco.

Se avete qualche esperienza con file di I/O in TI Basic o Extended BASIC allora vi accorgerete che in particolare nel preparare l'apertura di un file, il TurboForth è molto diverso. Tuttavia, una volta che il file è stato aperto, è tutto abbastanza naturale e non differisce molto dal Basic.

Il codice illustra la procedura e descrive tutto quello che c'è da sapere e da fare per aprire un file DV80 in input. Il codice è dettagliatamente commentato. Serguirà una versione ripulita del codice senza i rispettivi commenti.

Se non si conosce nulla del Forth, il migliore sito per iniziare è qui sotto:

http://turboforth.net/about_forth.html

Il link qui sopra inizia dai primi elementi. Poi, dopo averli letti, date un'occhiata al tutorial su TurboForth.net:

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

Le esercitazioni iniziano fin dai primi principi di base e via via avanzano in complessità.

Così qui di seguito trovate il codice per aprire un file DV80 e visualizzarlo:
(***Non essendo esperti del linguaggio FORTH e per non incappare in traduzioni sballate si riporta il testo originale non tradotto.)***

Le Linee di testo che sono precedute dal simbolo "****" sono *commenti* e vengono ignorate dal 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
;
```

Il Codice sopra riprtato può essere copiato e incollato in una finestra dell'emulatore Classic 99 ed eseguito immediatamente. Sarà necessario dare un nome di un file esistente su uno dei tuoi dischi. Ti Basterà digitare '**typefile**' per avviare il programma.

RESOURCES

Informazioni

Per contattarmi non esitate a visitare il mio sito e fare clic sulla scheda '**Contatti**'.

Argomenti per la Newsletter

Se volete partecipare alla stesura di questa newsletter e fornire argomenti per questa newsletter vi prego di contattarmi tramite il mio sito web.

Siti

Qui di seguito trovate le risorse in una manciata di siti che supportano i computer TI-99/4A e/o Geneve 9640. Non è certamente un elenco completo. Questa sezione sarà inclusa e aggiornata in tutte le prossime newsletter.

Siti Web / siti FTP

http://www.ti99iuc.it
http://www.atariage.com
http://shift838.wix.com/shift838
http://www.99er.net
http://www.harmlesslion.com
http://www.mainbyte.com
http://www.ninerpedia.org/
http://www.ti99-geek.nl/
http://www.turboforth.net/
ftp://ftp.whtech.com

new http://www.ti99hof.org/index.html
new http://www.ti99ers.org/unsung/

new http://ti99ers.org/modules/Inspire/remember.htm
contiene tutti gli storici TI-99ers che sono deceduti.

Lista Gruppi Yahoo

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

BBS attive

HeatWave BBS

Accesso: Dial-Up e Telnet
Sistema: Geneve 9640
Software: S&T BBS Software
Località: Arizona
Contenuto: biblioteche di file TI e Geneve, messaggi di base, porte giochi ed e-mail.
Telnet: www.heatwavebbs.com port 9640 dialup: 602-955-4491 @ 8-N-1

The Reef Hidden

Accesso: Dial-Up
Sistema: TI-99/4A modificato
Software: S&T BBS Software
Località: New York
Contenuto: biblioteche di file TI e Geneve, messaggi di base, porte giochi ed e-mail.

The Keep

Accesso: HTTP e Telnet
Sistema: Pentium 4 con sistema operativo Windows 2000
Software: Worldgroup BBS Software (fino a 256 connessioni utente)
Località: Tigard, Oregon
Contenuto: biblioteche di file TI e Geneve, messaggi di base, porte giochi, multi-utente e giochi multiplayer ed e-mail.
Telnet: www.thekeep.net porta 23 Web browser per http://www.thekeep.net
The Keep dispone di librerie di file TI, messaggi di base, e-mail, giochi porte, multi-utente e giochi multiplayer. The Keep ha anche una linea modem collegata con tutti coloro che desiderano contattare The Hidden Reef BBS da internet attraverso The Keep.
Semplicemente Telnet alla www.thekeep.net sulla porta 23, accedi a The Keep e quindi digita **/GO DIALOUT** nel menu principale, quindi D1 per la composizione verso The Hidden Reef. E' molto semplice.

Venditori

SHIFT838 – Fornisce componenti TI usati come li ha acquistati. Controllate spesso cosa ho disponibile. Un sacco di articoli possono essere riutilizzati da altri utenti TI.

Arcade Shopper - fornisce attrezzature TI vecchie e nuove, aggiornamenti e nuove piste PCB a www.arcadeshopper.com

Centri di riparazione

Richard Bell

Riparazioni disponibili su base limitata, si prega di contattare Richard a swim4home@verizon.net per conoscere i tempi di attesa prima di inviare qualsiasi componente da riparare

Tim

Riparazioni su hardware Myarc disponibili su base limitata. Contattare Tim a insane_m@hotmail.com per itempi di attesa o per richiedere il servizio.