2025/11/27 20:43 1/8 VDIP/USB am TINY

# tiny\_vdip.htm

## VDIP am TINY: Der USB-Stick-Anschluss

von Uwe Nickel

{{vdip1-150.jpg?150x101|Click for enlarged photo}}vdip s. http://www.vinculum.com/prd vdip1.html

Den Artikel und alle Unterlagen gibt es als Download Paket (2,5 MByte!)

Und nun kommt Uwe Nickel zu Wort:

## TINY und der USB-Stick Experimente mit Vinculum Vdip

### **Erfahrungsbericht**

## **Eingrenzung:**

Dieser Artikel soll keine kochrezeptartige Nachbauanleitung mit fertiger Leiterplatte und Software sein, sondern MĶglichkeiten aufzeigen am TINY mit relativ geringem Aufwand Vdip-Bastelexperimente durchzufļhren, ist also in diesem Sinne ein umfĤnglicher Erfahrungsbericht meiner eigenen Entwicklungen am TINY, die nun letztendlich zu einer fļr mich dauerhaften LĶsung als Unikat fļhrten. Ich stelle die dazu vorhandenen Unterlagen Interessenten frei zur Verfľgung.

Der Hardwareaufbau zwecks Experimente fand auf Uni-Leiterplatte statt, also gibt es auch kein Leiterplattenlayout. Die in mein bestehendes System, das vollstĤndig auf Uni-Leiterplatte aufgebaut ist, integrierte Erweiterung erfolgte in gleicher Art. (s.Bild)

Da ich nach wie vor in reinem Hexcode am Tiny mit dem Prog-Kommando und einigen eigenen Kommandoerweiterungen die Software entwickle, kann ich keinen  $\hat{A}$  fertig kommentierten $\hat{A}$  Quellcode in  $\hat{h}$  heren Programmiersprachen als Datei liefern, zumal sich meine Softwarerealisierung auf etliche Routinen meiner Betriebssystemerweiterung st $\hat{A}$  tzt.

Ein aktuelles Speicherdump des Gesamtpakets (Bereich ab 0000h  $\hat{A}_{\square}$  5FFFh) stelle ich auf Nachfrage (u.nickel@lycos.com) jedoch gern zur Verf $\hat{A}_{\square}$ gung. Bedienung usw. k $\hat{A}_{\square}$ nnen dann individuell gekl $\hat{A}_{\square}$ rt werden, das w $\hat{A}_{\square}$ rde im Einzelnen den Rahmen dieses Artikels sprengen. Zu beachten: Die Software hat experimentellen Charakter und ist auch noch behaftet mit etlichen Bugs.

Zum  $\hat{A}$  Nur mal angucken $\hat{A}$  I $\tilde{A}$  uft sie mit gewissen Einschr $\tilde{A}$  nkungen auch im JTC-Emulator von Jens M $\tilde{A}$  ller  $\hat{A}$  Dank an dieser Stelle f $\tilde{A}$  seine Entwicklungsarbeit!

Wer sich abgescannte, bleistiftbeschriebene A4-BlĤttern des Listings als Quellcode (mit nicht ganz normgerechter Mnemonic) antun mĶchte, kann diese Unterlagen gerne von mir erhalten. Ich gehe davon aus, dass der Leser des Artikels sich die hier erwĤhnten DatenblĤtter und sonstigen Unterlagen zum Modul von der Webseite der Firma selbst downloaded.

Auf bildliche Darstellungen oder Textauszýge aus o.g. Quellen Artikel verzichte ich, um ggf. vorhandene Copyright- bzw. Urheberrechte etc. nicht zu verletzen.

## Notwendige Hilfsmittel bei der Realisierung:

Im Projekt werden GALs eingesetzt, also ist ein Galbrenner nebst Software notwendig, oder die Möglichkeit, sich welche brennen zu lassen. Ich verwende einen GALBLASTER -Selbstbau incl. der Software und zum Entwickeln des JEDEC -Files das kostenlose WinCupl. Weiterhin ist eben nur das Â $\$ normaleÂ $\$ Bastel- Equipment erforderlich. Der Schwierigkeitsgrad des Hardwareaufbaus ist also als nicht sonderlich hoch einzusch $\$ Aztzen und d $\$ A $\$ 4rfte demzufolge auch f $\$ A $\$ 4r unge $\$ A $\$ 4btere TINY-Fans bei Interesse machbar sein.

## Vorüberlegungen und Grundlagen- Experimente:

Mein erstes Vinculum-Modul war/ist ein VDRIVE, also nicht das im Bastelbereich gern benutzte Dip-Modul. Damit standen mir vorerst nur wahlweise die UART oder SPI-Schnitstelle zur Verfýgung. Um einen ersten einfachen Kontakt zwischen TINY und VDRIVE herzustellen, wollte ich mit der seriellen Schnittstelle, wie ich sie auch für Druckeransteuerung (K6304) und Datenübertragung verwende, beginnen. Bisher lief die serielle Schnittstelle, die analog zum Originalartikel in der Zeitschrift Ju+Te aufgebaut ist, bei mir nur mit 1200 Baud. Der Vinculum arbeitet standardmÃxÃ[ig mit 9600/8/n/x und Hardwarehandshake per CTS/RTS. Prinzipiell ist per Vinculum-Tools auch eine VerÃxnderung der Firmware-Baudrate und Flusskontrolle komfortabel möglich, darauf habe ich jedoch verzichtet um nicht bei eventuell unbemerkten Misserfolg der Ã[nderung weitere Fehlerquellen im Versuchsaufbau einzubauen. Auch auf das Einspielen der aktuellsten Firmware habe ich verzichtet, obwohl es allerorts empfohlen wird. Nach eigener Recherche auf der Vinculum- Seite die Update-Versionshistorie betreffend, war festzustellen, dass für erste Schritte beim Kennenlernen des Moduls keine wesentlichen VerÄxnderungen der Firmware passiert sind. So galt es also festzustellen, ob der TINY 9600,8,n,x schafft, ohne Verwendung der UART, also mit rein softwarebasierter Ausgabe/Empfang über allgemeine Portpins.

XAls erstes Experiment habe ich da einfach die seriellen Leitungen des TINY an einen PC mit serieller Schnittstelle (virtuelle USB - COM) angeschlossen und per Terminalprogramm Bytes hin- und hergeschickt bei unterschiedlichen ̸bertragungsraten. Natürlich war eine Pegelanpassung notwendig, die erfolgte per MAX 232 in Standardbeschaltung! Die Kopplung PC-TINY erwies sich per Terminalprogramm als recht problemlos und es zeigte sich, dass die benötigte Geschwindigkeit erreichbar ist, auch ohne UART, Interruptbetrieb etc. Klar, dass dabei der Bildschirminterrupt des TINY ausgeschaltet sein muss, also während Senden oder Empfangen von einzelnen Bytes keine Bildschirmdarstellung erfolgen kann.

Folgende Routine war dabei Empfangsroutine des TINY (RP beliebig, ich verwende Registergruppe 5,  $R\tilde{A}^{1}/_{c}$ ckgabe des Zeichens in R5D Stoppbits werden nicht ausgewertet):

```
E6 5C F0
                LD %5C, #%F0
                                 Zeitkonst. für 1k2Bd, Einsprungadr. für
Empfang 1k2 Bd
    8B 03
                 JR LBL1
                LD %5C, #%1E
    E6 5C 1E
                                  Zeitkonst. für 9k6Bd, Einsprungadr. für
Empfang 9k6 Bd
LBL1
        70 E9
                                      Einsprungadresse für wählbare Geschw.
                      PUSH R9
    70 E8
                 PUSH R8
                                  Zeitkonstantenübergabe in R5C
    B<sub>0</sub> 5D
                  CLR %5D
    56 03 DF
                AND %3, #%DF
                                 RTS=L setzen
    8F
              DΙ
                         Interruptsperre
LBL2
        76 03 01
                                  Startbit abwarten
                     TM %3, #1
    EB FB
                  JRNZ, LBL2
    8C 08
                  LD R8, #8
                               Bitzahl
LBL5
        98 5C
                                     Zeitkonstante nach r9
                      LD R9, %5C
```

2025/11/27 20:43 3/8 VDIP/USB am TINY

```
LBL3
        9A FE
                      DJNZ R9, LBL3
                                        Zeitschleife
    66 03 01
                 TCM %3, #1
                               Bit = H ?
    EB 03
                  JRNZ, LBL4
                 OR %5D, #%80
    46 5D 80
LBL4
        E0 5D
                      RR %5D
    8A F0
                  DJNZ R8, LBL5
                                    Wiederholung bis zum 8.Bit
    46 03 20
                 OR %3, #%20
                                 RTS=H setzen
    50 E8
                  POP R8
    50 E9
                  POP R9
    9F
              ΕI
    ΑF
              RET
```

Die folgenden Zeitkonstanten in R5C habe ich experimentell ermittelt:

Baud	Konstante Hex
1200	F0
2400	78
4800	3C
9600	1E
19200	0D

Die verwendete Senderoutine ist eine angepasste Version der in der Ursprungsliteratur (Zeitschrift Ju+Te) angegebenen Routine zur Druckeransteuerung, die ich genauso umgeschrieben habe, dass ich in R5C jeweils eine Zeitkonstante  $\tilde{A}^{1/4}$ bergebe.

 $W\tilde{A}$ xhrend bei Verwendung des PC-Terminals zur ordentlichen Daten $\tilde{A}$ xbertragung (Halbduplex) die 3 Leitungen (TxD, RxD, 1xHandshake) ausreichend sind, musste jedoch f $\tilde{A}$ xr den VDRIVE-Anschluss eine weitere Portleitung als 2. Handshakesignal herhalten.

#### TINY und Vinculum seriell

Obige Schaltung reicht f $\tilde{A}^{1/4}$ r die Kommunikation zwischen TINY und Vinculum also nicht aus. Es ist voller RTS-CTS Handshake (da ich ja keine andere Flusskontrolle per Tool einstellen wollte), nach folgender Verdrahtung bei mir realisiert.

Ursache ist logischerweise der Vollduplex-Datentransfer, den der Vinculum seinerseits macht, wĤhrend ja der TINY keinen Sende und Empfangspuffer hat (bei der von mir angestrebten UART- und interruptfreien Betriebsweise!).

Also Â[verpasstÂ] der gemächliche TINY regelmäÃ[ig Zeichen des Vinculum, wenn er diesem nicht mitteilt, dass der doch bitte warten möge! Für meine Experimente opferte ich somit einfach eine weitere Portleitung. Pegelwandlung zwischen TINY und Vinculum ist dabei nicht notwendig- Vdrive-Modul liefert ja TTL-kompatible Signale, was den Anschluss schon sehr vereinfacht. Nach Anpassung der Empfangsroutine konnte ich erste Gehversuche auch sofort machen. Die dazu erarbeitete Minimalsoftware funktionierte einfach so:

Tastatureingaben am TINY werden 1:1 Byte fÃ $\frac{1}{4}$ r Byte zum VDRIVE Ã $\frac{1}{4}$ bertragen (und natÃ $\frac{1}{4}$ rlich auf dem Bildschirm dargestellt) bis mit Enter 0D abgeschlossen wird. Dann wird sofort auf Antwort vom VDRIVE gewartet, jedes empfangene Zeichen auf dem Bildschirm dargestellt, bis die Ausgabe beendet ist. Und dann da capo al fine. Also einfaches Wechselspiel Â $\Box$ Kommando  Antwort  Kommando Â $\Box$ Â $\Box$ Â $\Box$ 

Ging so halbwegs, war aber insgesamt nicht so ganz berauschend. Dabei tauchte softwareseitig n $\tilde{A}$  $\mu$ mlich ein Problem auf, das sich eigentlich auch bei allen anderen Anschlussarten wiederholt: Wann ist denn nun eigentlich ein Ausgabezyklus des Moduls (also  $\hat{A}$  $\mu$ ntwort $\hat{A}$ ) beendet? Woran ist das erkennbar? Bei Nachfragen an Vinculum-Spezialisten im Internet lautete die lapidare Antwort eigentlich immer: Wo ist denn da das Problem, nat $\hat{A}$  $\mu$ rlich mit Prompt gefolgt von  $\hat{A}$ 0DH $\hat{A}$ - Enter. Ganz so stimmt das aber nicht! Denn nur ein erfolgreiches Kommando erzeugt als Antwort o.g.  $\hat{A}$  $\mu$ 0 kgabeende. Bei Fehler kommt die entsprechende Meldung seitens des Moduls gefolgt von nur Enter! Und nur Enter andererseits kommt auch bei einigen Antworten zwecks Ausgabeformatierung  $\hat{A}$ mittendrin $\hat{A}$ , siehe DIR o. $\hat{A}$  $\mu$ 2.

Eine getimte Abfrage des Vdrive brachte auf die Schnelle keine wesentliche Verbesserung des Gesamtverhaltens. Je nach DatentrĤger im Vdrive entstanden bei grĶÄ□eren Datenmengen unterschiedlich lange Â□DenkpausenÂ□ des Moduls. Das im Befehlssatz des Vinculum vorhandene Â□EÂ□-Kommando (Echo) kann hier jedoch als Synchronisationssignal zur Erkennung eingesetzt werden. Zusammen mit der Tatsache, dass ich bei serieller Realisierung mir ja entweder die vorhandene serielle Schnittstelle blockiere, oder mir eben eine weitere bauen muss, lieÃ□ mich dann von einer seriellen Lösung doch Abstand nehmen.

#### Ist vielleicht SPI besser?

Genau das hab ich dann nicht mehr probiert denn das hätte auch den Aufbau einer softwarebasierten Schnittstelle mit entsprechenden Portanschlüssen bedeutet. Aber allgemein an SPI war ja mein Interesse sowieso schon lange geweckt und unabhängig vom Vinculum wird das eine der nächsten Basteleien werden.

#### Iso doch Parallel?!

Auch wenn 9600 Byte/s ja nicht schlecht sind, die letztendliche Realisierung habe ich dann doch aus o.g. Gründen nicht seriell gemacht. Inzwischen lag auch das VDip-Modul vor mir, somit waren parallele Experimente möglich. Nach kurzer Recherche im Internet wurde ich auf einigen Seiten bzgl. des Parallelbetriebs auch fündig. Insbesondere die für den KC 85-System verwendete Schaltung war f $\tilde{A}\frac{1}{4}$ r erste  $\tilde{A}$  berlegungen sehr hilfreich. Der PIO-Aufwand  $\hat{A}$  f $\tilde{A}\frac{1}{4}$ r das KC-System vollkommen richtig - schreckte mich jedoch zuerst in Gedanken an das notwendige LA¶ten an meinem Aufbau, zumal der Platz für Erweiterungen auf der Leiterplatte langsam eng wird. AuÃ∏erdem stand die Frage im Raum, womit die Ports realisiert werden sollten. Wenn, dann wĤre wahrscheinlich nur was in Standard-Logik in Frage gekommen, denn eine Z80PIO anstricken scheitert schon mal an der Taktfrequenz (ich lasse meinen Aufbau manchmal auch mit 16MHz laufen) und andere Spezialbausteine lagen mir nicht vor. Erstes Betrachten des Datenblattes zeigte ja, dass es kein CE-Signal gibt. Also Anschluss über irgendein E/A-Tor zwingend notwendig!? Beim zusätzlichen betrachten des Timingdiagramms des Vdip drĤngte sich folgende Frage auf: Was machen eigentlich die Datenpins des Vdip, wenn sowohl Schreib- als auch Lesesignal inaktiv sind? Â\[Blackbox\hat{A}\[]- Testen ergab, sie verhalten sich elektrisch wie EingĤnge, obwohl natürlich dann datentechnisch im VDip nichts passiert.

#### Demzufolge also die Idee:

- Ich schlieà e das Modul mit den Datenpins direkt an den Datenbus, versuche somit zusÃxtzlichen Hardwareaufwand für ein 8bit- E/A-Tor zu sparen,
- erzeuge Â□lokaleÂ□ Schreib- bzw. Lesesignale, d.h. die nur bei der entsprechend für das Modul vorgesehenen Adresse(n) aktiv werden, per Gal leicht möglich.
- 3. à ber ein Mini-Eingabeport es kà nnten auch x-beliebige schon vorhandene Portleitungen sein, werden die beiden Statusleitungen (RxF, TxE) entsprechend abgefragt.

2025/11/27 20:43 5/8 VDIP/USB am TINY

{{tiny\_v4.JPG}}Dazu reicht dann eine GAL völlig aus, als Minimalvariante, wie im Bild links ersichtlich. Aber da ich ohnehin noch einige Erweiterungsideen habe, ist es eher angebracht etwas mehr Aufwand zu treiben und einige weitere Ports zu organisieren. So sieht die realisierte Schaltung dann entsprechend aus.

Die GAL-Files realisieren auch hier die Ansteuerung. Die Schaltung wurde dabei f $\tilde{A}\frac{1}{4}$ r Polling-Abfragen (/TxE,RxF) ausgelegt. Interruptbetrieb, wie in der Schaltung f $\tilde{A}\frac{1}{4}$ r den KC, habe ich nicht vorgesehen.



Fżr das Schreiben und Lesen von Bytes mit dem VDip ist zuerst der jeweilige Status auf der Statussignal- Adresse zu lesen und in Auswertung der Signale, die entsprechende Aktion auf der Daten-Adresse durchzuführen.

## Achtung! Wichtige Ergänzung im Nachhinein fþr zuverlässige Funktion:

Ich habe ganz viel Zeit bei der Suche nach einem vermeintlichen Fehler in Hard- bzw. Software vergeudet, der sich wie folgt  $\tilde{A}$  $\alpha$ u $\tilde{A}$  $\beta$ erte: Nachdem die Software fertig geschrieben war, inklusive der M $\tilde{A}$  $\beta$ glichkeit vom Stick zu booten, stellte sich heraus, dass Leseoperationen vom Stick bzw. Vdip unregelm $\tilde{A}$  $\alpha$  $\beta$ gig auftretend, nicht zuverl $\tilde{A}$  $\alpha$ ssig funktionierten. Schreiboperationen auf den Stick waren jedoch immer fehlerfrei. Es klappte aber eben das Laden von Dateien bzw. das booten vom Stick oft nicht. Durch im Ausschlussverfahren gef $\tilde{A}$ 1/4hrte Experimente glaube ich das Problem nun zu kennen, habe es zumindest erfolgreich beseitigt, Messungen kann ich dazu nicht nachweisen, es fehlt dazu das Equipment:

Das Vdip-Modul ist eigentlich ja 3,3Volt-Logik. Die Pins sind lediglich 5Volt-tolerant. Wenn der Datenbus des Z8-Systems, wie in meinem Fall durch ungünstige Leitungslänge und viele angeschlossene weitere Schaltkreise Â□stark belastetÂ□ ist, dann tritt oben genanntes Problem auf, das Vdip kann also scheinbar nicht bzw. nicht schnell genug ordentlich Pegel liefern. Ist die Busbelastung geringer, dann funktioniert es ohne Probleme. Zur Abhilfe habe ich oben gezeigte Schaltung um einen LS 245 erweitert, der als Busdriver zwischen Datenbus und Datenpins des VdipÂ□s liegt. Damit hat dann das Modul nur eine LS-Last zu treiben und das Problem tritt nicht mehr auf. Leider geht damit der Vorteil ein 8-bit Â□Tor in der Ansteuerung zu sparen verloren, aber das Ergebnis rechtfertigt den Aufwand.

#### **Und die Software?**

Im  $\hat{A}$  Probierfall $\hat{A}$  also wieder einfach  $\tilde{A}$  ber Tastatur eingegebene Werte solange an das Modul weitergeben incl. Enter als Abschluss, dann Bytes lesen und auf dem Bildschirm wiedergeben. Das reicht ja dann schon um die Kommunikation zu testen.



Weiter Eindrücke unter: http://picasaweb.google.de/unick59

Soll eine vernļnftige Interaktion mit dem Benutzer erfolgen, ist natļrlich ein wenig mehr Aufwand angebracht.

Meine Realisierung des Ablaufes des Programms, das nun einen eigenen Menupunkt im TINY-Startmenue bildet, habe ich  $\hat{A} \cup VDOS 1.x\hat{A}$  getauft. In der Grundroutine erfolgt die Statusabfrage des VDip nicht  $\hat{A} \cup VD$  wartend $\hat{A} \cup VD$ , sondern dynamisch. Nach Abfrage wird bei Ausgabe-Bereitschaft des Moduls

eben Bildschirmausgabe eines Zeichens gemacht, wenn keine Bereitschaft vorliegt nach Tastaturabfrage eben wieder Bereitschaftsabfrage usw. Damit wird realisiert, dass z.B. auch das Entfernen des Sticks aus dem Modul erkannt wird. Die Erkennung des Antwort-Ausgabeendes (s.beschriebenes Problem weiter oben) ist im Parallelmode unkomplizierter, da das entsprechende Statussignal /RxF dann dauerhaft H-Pegel führt, da keine neuen Daten mehr im Puffer anliegen. Aber Achtung, H-Pegel ist auch immer dann, wenn während einer laufenden Datenausgabe eben mal gerade keine neuen Daten anliegen, weil das Modul beschäftigt ist.

Etwas, zum Teil auch akademischen, Aufwand habe ich bei der Anpassung der Bildschirmausgaben getrieben. Hintergrund war die Ä\berlegung, dass bei Darstellung von Verzeichnissen etc. die Zeilenzahl, die auf dem TINY-Bildschirm dargestellt werden kann, ja ganz schnell erreicht bzw. \tilde{A}\frac{1}{4}berschritten wird, also Scrolling erfolgt. Damit geht dann logischerweise eben auch ggf. Information f\tilde{A}\frac{1}{4}r den Benutzer verloren. Deshalb habe ich M\tilde{A}\tilde{g}lichkeiten eingebaut die Ausgabe anzuhalten (per Druck auf bestimmte Taste). Durch Bet\tilde{A}\tilde{x}tigung bestimmter Tasten(kombinationen) werden dann Ausgabestoppbedingungen eingestellt, die ab dann bis zur n\tilde{A}\tilde{x}chsten \tilde{A}\bigcap nderung gelten:

Bei Tastendruck  $\hat{A} \square$  am Prompt $\hat{A} \square$  wird sofort die erste Taste ausgewertet, so ihr eine bestimmte Funktion in einer Tabelle zugeordnet ist. Z.B.

- Sh + ET Â

  ☐ Verlassen des Programms,
- ET neues Prompt
- / Direktausgabe der Zeichen an das Vdip
- Sh +Clr Befehlspuffer auf Bildschirm ausgeben (s. F3 bei DOS)
- ! in den ECS-Modus schalten, falls Vdip im SCS-Modus ist.

Wird die 1. Taste nicht als  $\hat{A} \square$  gelistet $\hat{A} \square$  erkannt, erfolgt einfach weitere Bildschirmausgabe bis mit Enter abgeschlossen wird. Dann wird der zwischen aktueller Cursorposition und letztem  $\hat{A} \square$  davor liegenden $\hat{A} \square$  Prompt liegende Text als Kommando, von f $\hat{A}^1$ /4hrenden und anh $\hat{A}$ mngigen Leerzeichen ges $\hat{A}$ mubert, in den Befehlspuffer  $\hat{A}^1$ /4bernommen. Also so ein wenig a la Full-Screen-Editor. Ist nun das erste Zeichen im Puffer ein Punkt  $\hat{A} \square . \hat{A} \square$ , so wird der gesamte String ohne weitere Aktionen direkt an das Vdip gegeben und auf Antwort per Bildschirmausgabe gewartet in der Hauptschleife. Alle anderen Befehlsstrings werden vom TINy erst mal analysiert: Vom Stringanfang bis zum ersten Leerzeichen oder Enter wird die Zeichenkette mit einer Tabelle von Befehlen verglichen. Wird kein gelisteter Befehl erkannt, geht der String zum Vdip. Entweder ist es ein dem Vdip bekannter  $\hat{A} \square$  interner $\hat{A} \square$  Befehl mit richtiger Syntax, oder es erfolgt Fehlermeldung und man ist wieder in der Hauptschleife mit neuem Prompt.

Damit lassen sich erst mal alle Grundfunktionen des Moduls nutzen, ein Speichern oder Laden von Daten ist nun aber kein einzelner Befehl sondern setzt sich ja aus mehreren Schritten zusammen (Ķffnen der Datei, ggf. Pointer setzen, Bytes lesen bzw. Schreiben, Datei schliessen...) Genau diese Abfolgen sind die Befehle, die Vdip- extern in der TINY-Befehlstabelle gespeichert sind und also dann sozusagen, wie ein Macro abgearbeitet werden.

An dieser Stelle ergab sich die Frage nach dem Dateiformat der abzuspeichernden Daten. Ich habe prinzipiell unterschieden zwischen einem Headerlosen Format und einem Dateiformat in dem ein Kopf mit zusĤtzlichen Informationen mit abgespeichert wird. Damit wird eine derartige Datei um 10H Bytes lĤnger und ich speichere die Information zur Herkunft der Daten -Anfangsadresse, LĤnge, Speicherbank- und Page (da mein TINY dafļr getrennte Speicherbereiche hat, Stichwort P34-Einbeziehung in Adressdekodierung), sowie eine Startadresse ab. Ob die Datei einen Header hat, oder nicht, wird durch die Dateinamenserweiterung verifiziert.

2025/11/27 20:43 7/8 VDIP/USB am TINY

Momentan kennt mein TINY nun folgende:

- \*.dmp, \*.bin Â□ ohne Header
- \*.hex Â□ mit Header, aber keine Startadresse
- \*. Exe, \*.com, \*.bas, \*.fth, \*.ox Â□ Header incl. Startadresse.

Unbekannte Dateinamenserweiterungen werden als Headerbehaftet angesehen.

F¼r Datei-Speicheroperationen habe ich 3 Befehle implementiert:

- SD Â□ interaktives Speichern aus dem externen Datenspeicher,
- SP- analog SD für den aktuellen Programmspeicher
- S\_Dateiname.ext Â
   ☐ setzt bei bekannten Erweiterungen, wie bas (BASIC), fth (FORTH) gÃ
   ¼Itige Vorgabewerte fÃ
   ¼r Ablageadresse, Startadresse etc.



Analog existieren auch 3 Ladebefehle.

Bekannte Datei(-typen), die  $\tilde{A}\frac{1}{4}$ ber ein Startadresse verf $\tilde{A}\frac{1}{4}$ gen und per L\_Dateiname.ext geladen werden, werden dann ausgef $\tilde{A}\frac{1}{4}$ hrt. (das JUTE-Forth in 5 Sekunden geladen und gestartet ist schon ein Vergn $\tilde{A}\frac{1}{4}$ gen!)



L Dateiname.ox startet dann also auch ein auf dem Stick vorhandenes Betriebssystem.

Damit ist nun der Weg nicht mehr weit einfach dem Tiny per einzelnen Tastendruck im Grundmenue zu sagen, nun lade mal schnell das Betriebssystem TINY.OX vom Stick, wenn du es findest, und starte dich neu  $\hat{A}$  bei mir jetzt Kommando  $\hat{A}$  m Grundmenue. Und nur ein kleiner Schritt weiter ist logischerweise diesen Vorgang gleich beim Kaltstart zu erledigen und nur, wenn kein Stick vorhanden, oder die benannte Datei fehlt, aus dem ROM zu starten. Also booten vom Stick.

Mit diesen Möglichkeiten arbeite ich nun schon eine Weile mit viel SpaÃ∏, habe mich inzwischen neuen TINY\_Projekten zugewandt und möchte den Komfort der schnellen Datenspeicherung nicht mehr missen.

VGA-Anschluss und Pc  $\hat{A}$  Tastaturanschluss sind in der Zwischenzeit prinzipiell realisiert. Nach Beseitigung der letzten Bugs, werde ich dar $\hat{A}$  ber berichten.

## Hinweise für Verwendung des ROM-ImagesÂ□ Unterschiede zum Standard-TINY :



weitere Bilder unter: http://picasaweb.google.de/unick59

Das ROM-Image läuft definitiv nicht ad hoc in einer Standard-Hardwareumgebung! D.h. es läuft schon, nur es sind keine Tastatureingaben möglich, da ich zwecks Schaffung eines möglichst durchgehenden RAM-Bereiches Alles, was I/O-Operationen anbelangt, in die obersten 2k Â $_$ gepacktÂ $_$ habe. In genau diesem Â $_$ System-RAMÂ $_$ -Bereich liegt also wie gehabt auch der Bildschirmspeicher, dort sind die BIOS-Zellen, die RTC, die Adressen des Vinculum, Pufferspeicher fÃ $_$ 4r Flashroutinen etc... FÃ $_$ 4r das Ausprobieren im Emu von Jens MÃ $_$ 4ller spielt es keine Rolle, deshalb empfehle ich fÃ $_$ 4r das Â $_$ 6mal schnell anguckenÂ $_$ 6 den Emu.

Die Software hat noch alpha-Status, also etliche Bugs, viel internen MüII. So die konkrete VDip-Hardware nicht vorhanden ist, wird die Routine abgebrochen. Deshalb Iäuft das auch nicht im Emulator.

In meinem System sind Daten und Programmspeicher getrennt (P34 =/DM). Bei den meisten Menupunkten und Befehlen ist deshalb die EinstellmĶglichkeit für eine Speicherbank und Â□seite vorgesehen. Das bleibt im Emu und im Standard-Tiny also wirkungslos. Das reale System verfügt Ã⅓ber Â□Bios-MerkzellenÂ□, untergebracht in einer Echtzeituhr. Davon ausgehend werden beim Starten bzw. nach Reset erst etliche Such-und Ã□berprÃ⅓fungsvorgänge ausgeführt um ggf. ein im Datenspeicher oder im EPROM des Programmspeichers abgelegtes Betriebssystem zu laden. Ich habe, wie schon oben erwähnt, durchgehenden RAM-Bereich im Programmspeicher, lade das Betriebssystem aus einem Festwertwertspeicher (EPROM, FLASH, Zeropower-RAM...) in den RAM. Das Starten dauert also länger. Im Startbildschirm erscheint Datum und Uhrzeit. Im Emu hat auch das sonst keine lauffähigkeitsbedingte Auswirkung. Da ich per Echtzeituhr Â□Steuerung einen blinkenden Cursor habe, kann im Emu es vorkommen, dass er verschwunden ist.

Seit geraumer Zeit benutze ich schon einen Zilog als CPU. Ich habe mir das R-Kommando neu geschrieben um auch an alle Register und alle Speicherbereiche ranzukommen. Ich musste feststellen, dass das mit der Originalroutine nicht ganz klappt. Sie war ja auch fÃ $\frac{1}{4}$ r den 8830 gedacht. Altes Â $\frac{1}{4}$ RÂ $\frac{1}{4}$  ist bei mir Â $\frac{1}{4}$ HÂ $\frac{1}{4}$ . Der Aufruf der einzelner Befehle aus dem Hauptbildschirm erfolgt durch den zugeordneten Buchstaben, wie gewohnt. Zwischen den Menuseiten wird mit + bzw. Â $\frac{1}{4}$  umgeschaltet. Der Start eines Programms erfolgt aus dem Grundmenue heraus mit Â $\frac{1}{4}$ CxxxxÂ $\frac{1}{4}$ Dabei ist xxxx- Adresse in Hex. Damit wird das umstÃ $\frac{1}{4}$ mdliche Prog aufrufen, G adr eingeben, L adr eingeben verkÃ $\frac{1}{4}$ rzt, ist aber auch mÃ $\frac{1}{4}$ glich.

Für alle weiteren Erklärungen Â∏ bei Interesse einfach nachfragen!

Uwe Nickel, 06/2009

From:

https://hc-ddr.hucki.net/wiki/ - Homecomputer DDR

Permanent link:

https://hc-ddr.hucki.net/wiki/doku.php/tiny/erweiterungen/vdip?rev=1278858422

Last update: 2010/07/10 22:00

