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 Â Experimente mit Vinculum Vdip

** Erfahrungsbericht**

Eingrenzung:

Dieser Artikel soll keine kochrezeptartige Nachbauanleitung mit fertiger Leiterplatte und Software sein, sondern MÃ¶glichkeiten aufzeigen am TINY mit relativ geringem Aufwand Vdip-Bastelexperimente durchzufÃ¼hren, ist also in diesem Sinne ein umfÃ¤nglicher Erfahrungsbericht meiner eigenen Entwicklungen am TINY, die nun letztendlich zu einer fÃ¼r mich dauerhaften LÃ¶sung als Unikat fÃ¼hrten. Ich stelle die dazu vorhandenen Unterlagen Interessenten frei zur VerfÃ¼gung.

Der Hardwareaufbau zwecks Experimente fand auf Uni-Leiterplatte statt, also gibt es auch kein Leiterplattenlayout. Die in mein bestehendes System, das vollstÃ¤ndig 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 Âfertig kommentiertenÂ Quellcode in hÃ¶heren Programmiersprachen als Datei liefern, zumal sich meine Softwarerealisierung auf etliche Routinen meiner Betriebssystemerweiterung stÃ¼tzt.

Ein aktuelles Speicherdump des Gesamtpakets (Bereich ab 0000h Â 5FFFh) stelle ich auf Nachfrage (u.nickel@lycos.com) jedoch gern zur VerfÃ¼gung. Bedienung usw. kÃ¶nnen dann individuell geklÃ¤rt werden, das wÃ¼rde im Einzelnen den Rahmen dieses Artikels sprengen. Zu beachten: Die Software hat experimentellen Charakter und ist auch noch behaftet mit etlichen Bugs.

Zum ÂNur mal anguckenÂ lÃ¤uft sie mit gewissen EinschrÃ¤nkungen auch im JTC-Emulator von Jens MÃ¼ller Â Dank an dieser Stelle fÃ¼r seine Entwicklungsarbeit!

Wer sich abgescannte, bleistiftbeschriebene A4-BlÃ¤ttern des Listings als Quellcode (mit nicht ganz normgerechter Mnemonic) antun mÃ¶chte, kann diese Unterlagen gerne von mir erhalten. Ich gehe davon aus, dass der Leser des Artikels sich die hier erwÃ¤hnten DatenblÃ¤tter und sonstigen Unterlagen zum Modul von der Webseite der Firma selbst downloaded.

Auf bildliche Darstellungen oder TextauszÃ¼ge 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 MÃ¶glichkeit, 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 einzuschÃ¤tzen und dÃ¼rfte demzufolge auch fÃ¼r ungeÃ¼btere TINY-Fans bei Interesse machbar sein.

VorÃ¼berlegungen 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 VerfÃ¼gung. Um einen ersten einfachen Kontakt zwischen TINY und VDRIVE herzustellen, wollte ich mit der seriellen Schnittstelle, wie ich sie auch fÃ¼r Druckeransteuerung (K6304) und DatenÃ¼bertragung 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 standardmÃ¤Ãig mit 9600/8/n/x und Hardwarehandshake per CTS/RTS. Prinzipiell ist per Vinculum-Tools auch eine VerÃ¤nderung der Firmware-Baudrate und Flusskontrolle komfortabel mÃ¶glich, darauf habe ich jedoch verzichtet um nicht bei eventuell unbemerkten Misserfolg der Ãnderung 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 fÃ¼r erste Schritte beim Kennenlernen des Moduls keine wesentlichen VerÃ¤nderungen 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 Ã¼ber allgemeine Portpins.

Als 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 Ãbertragungsraten. NatÃ¼rlich 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 benÃ¶tigte Geschwindigkeit erreichbar ist, auch ohne UART, Interruptbetrieb etc. Klar, dass dabei der Bildschirminterrupt des TINY ausgeschaltet sein muss, also wÃ¤hrend Senden oder Empfangen von einzelnen Bytes keine Bildschirmdarstellung erfolgen kann.

Folgende Routine war dabei Empfangsroutine des TINY (RP beliebig, ich verwende Registergruppe 5, RÃ¼ckgabe des Zeichens in R5D Stoppbits werden nicht ausgewertet):

```
    E6 5C F0    LD %5C, #%F0    Zeitkonst. fÃ¼r 1k2Bd, Einsprungadr. fÃ¼r
Empfang 1k2 Bd
    8B 03        JR LBL1
    E6 5C 1E    LD %5C, #%1E     Zeitkonst. fÃ¼r 9k6Bd, Einsprungadr. fÃ¼r
Empfang 9k6 Bd
LBL1    70 E9        PUSH R9        Einsprungadresse fÃ¼r wÃ¤hlbare Geschw.
    70 E8        PUSH R8        ZeitkonstantenÃ¼bergabe in R5C
    B0 5D        CLR %5D
    56 03 DF    AND %3, #%DF    RTS=L setzen
    8F        DI        Interruptsperre
LBL2    76 03 01    TM %3, #1    Startbit abwarten
    EB FB        JRNZ, LBL2
    8C 08        LD R8, #8    Bitzahl
```

```
LBL5    98 5C        LD R9, %5C    Zeitkonstante nach r9
LBL3    9A FE        DJNZ R9, LBL3    Zeitschleife
    66 03 01    TCM %3, #1    Bit = H ?
    EB 03        JRNZ, LBL4
    46 5D 80    OR %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        EI
    AF        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 Ã¼bergebe.

WÃ¤hrend bei Verwendung des PC-Terminals zur ordentlichen DatenÃ¼bertragung (Halbduplex) die 3 Leitungen (TxD, RxD, 1xHandshake) ausreichend sind, musste jedoch fÃ¼r den VDRIVE-Anschluss eine weitere Portleitung als 2. Handshakesignal herhalten.

TINY und Vinculum seriell

Obige Schaltung reicht fÃ¼r 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, wÃ¤hrend ja der TINY keinen Sende und Empfangspuffer hat (bei der von mir angestrebten UART- und interruptfreien Betriebsweise!).

Also Â verpasstÂ der gemÃ¤chliche TINY regelmÃ¤Ãig Zeichen des Vinculum, wenn er diesem nicht mitteilt, dass der doch bitte warten mÃ¶ge! FÃ¼r 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 fÃ¼r Byte zum VDRIVE Ã¼bertragen (und natÃ¼rlich 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 Kommando 

Antwort  Kommando Â□Â□Â□

Ging so halbwegs, war aber insgesamt nicht so ganz berauschend. Dabei tauchte softwareseitig nÃ¤mlich ein Problem auf, das sich eigentlich auch bei allen anderen Anschlussarten wiederholt: Wann ist denn nun eigentlich ein Ausgabezyklus des Moduls (also Â□AntwortÂ□) beendet? Woran ist das erkennbar? Bei Nachfragen an Vinculum-Spezialisten im Internet lautete die lapidare Antwort eigentlich immer: Wo ist denn da das Problem, natÃ¼rlich mit Prompt gefolgt von Â□0DHÂ□ - Enter. Ganz so stimmt das aber nicht! Denn nur ein erfolgreiches Kommando erzeugt als Antwort o.g. RÃ¼ckgabeende. Bei Fehler kommt die entsprechende Meldung seitens des Moduls gefolgt von nur Enter! Und nur Enter andererseits kommt auch bei einigen Antworten zwecks Ausgabeformatierung Â□mittendrinÂ□, siehe DIR o.Ã¤.

Eine getimte Abfrage des Vdrive brachte auf die Schnelle keine wesentliche Verbesserung des Gesamtverhaltens. Je nach DatentrÃ¤ger im Vdrive entstanden bei grÃ¶Ã□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 LÃ¶sung doch Abstand nehmen.

Ist vielleicht SPI besser?

Genau das hab ich dann nicht mehr probiert denn das hÃ¤tte auch den Aufbau einer softwarebasierten Schnittstelle mit entsprechenden PortanschlÃ¼ssen bedeutet. Aber allgemein an SPI war ja mein Interesse sowieso schon lange geweckt und unabhÃ¤ngig vom Vinculum wird das eine der nÃ¤chsten Basteleien werden.

Iso doch Parallel?!

Auch wenn 9600 Byte/s ja nicht schlecht sind, die letztendliche Realisierung habe ich dann doch aus o.g. GrÃ¼nden nicht seriell gemacht. Inzwischen lag auch das VDip-Modul vor mir, somit waren parallele Experimente mÃ¶glich. Nach kurzer Recherche im Internet wurde ich auf einigen Seiten bzgl. des Parallelbetriebs auch fÃ¼ndig. Insbesondere die fÃ¼r den KC 85-System verwendete Schaltung war fÃ¼r erste Ã□berlegungen sehr hilfreich. Der PIO-Aufwand Â□ fÃ¼r das KC-System vollkommen richtig - schreckte mich jedoch zuerst in Gedanken an das notwendige LÃ¶ten an meinem Aufbau, zumal der Platz fÃ¼r Erweiterungen auf der Leiterplatte langsam eng wird. AuÃ□erdem stand die Frage im Raum, womit die Ports realisiert werden sollten. Wenn, dann wÃ¤re 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 Ã¼ber irgendein E/A-Tor zwingend notwendig!? Beim zusÃ¤tzlichen betrachten des Timingdiagramms des Vdip drÃ¤ngte sich folgende Frage auf: Was machen eigentlich die Datenpins des Vdip, wenn sowohl Schreib- als auch Lesesignal inaktiv sind? Â□BlackboxÂ□- Testen ergab, sie verhalten sich elektrisch wie EingÃ¤nge, obwohl natÃ¼rlich dann datentechnisch im VDip nichts passiert.

Demzufolge also die Idee:

1. Ich schlieÃ□e das Modul mit den Datenpins direkt an den Datenbus, versuche somit zusÃ¤tzlichen Hardwareaufwand fÃ¼r ein 8bit- E/A-Tor zu sparen,
2. erzeuge Â□lokaleÂ□ Schreib- bzw. Lesesignale, d.h. die nur bei der entsprechend fÃ¼r das Modul vorgesehenen Adresse(n) aktiv werden, per Gal leicht mÃ¶glich.

3. Ã□ber ein Mini-Eingabeport Â□ es kÃ¶nnten auch x-beliebige schon vorhandene Portleitungen sein, werden die beiden Statusleitungen (RxF, TxE) entsprechend abgefragt.

{{tiny_v4.JPG}}Dazu reicht dann eine GAL vÃ¶llig 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 fÃ¼r Polling-Abfragen (/TxE,RxF) ausgelegt. Interruptbetrieb, wie in der Schaltung fÃ¼r den KC, habe ich nicht vorgesehen.

FÃ¼r 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 durchzufÃ¼hren.

Achtung! Wichtige ErgÃ¤nzung im Nachhinein fÃ¼r zuverlÃ¤ssige Funktion:

Ich habe ganz viel Zeit bei der Suche nach einem vermeintlichen Fehler in Hard- bzw. Software vergeudet, der sich wie folgt Ã¤uÃ□erte: Nachdem die Software fertig geschrieben war, inklusive der MÃ¶glichkeit vom Stick zu booten, stellte sich heraus, dass Leseoperationen vom Stick bzw. Vdip unregelmÃ¤Ã□ig auftretend, nicht zuverlÃ¤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 gefÃ¼hrte 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 ungÃ¼nstige LeitungslÃ¤nge 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 Â□ProbierfallÂ□ also wieder einfach Ã¼ber 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 EindrÃ¼cke unter: http://picasaweb.google.de/unick59

Soll eine vernÃ¼nftige Interaktion mit dem Benutzer erfolgen, ist natÃ¼rlich ein wenig mehr Aufwand angebracht.

Meine Realisierung des Ablaufes des Programms, das nun einen eigenen Menupunkt im TINY-Startmenue bildet, habe ich Â□VDOS 1.xÂ□ getauft. In der Grundroutine erfolgt die Statusabfrage des VDip nicht Â□wartendÂ□, 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 fÃ¼hrt, da keine neuen Daten mehr im Puffer anliegen. Aber Achtung, H-Pegel ist auch immer dann, wenn wÃ¤hrend einer laufenden Datenausgabe eben mal gerade keine neuen Daten anliegen, weil das Modul beschÃ¤ftigt ist.

Etwas, zum Teil auch akademischen, Aufwand habe ich bei der Anpassung der Bildschirmausgaben getrieben. Hintergrund war die Ã□berlegung, dass bei Darstellung von Verzeichnissen etc. die Zeilenzahl, die auf dem TINY-Bildschirm dargestellt werden kann, ja ganz schnell erreicht bzw. Ã¼berschritten wird, also Scrolling erfolgt. Damit geht dann logischerweise eben auch ggf. Information fÃ¼r den Benutzer verloren. Deshalb habe ich MÃ¶glichkeiten eingebaut die Ausgabe anzuhalten (per Druck auf bestimmte Taste). Durch BetÃ¤tigung bestimmter Tasten(kombinationen) werden dann Ausgabestoppbedingungen eingestellt, die ab dann bis zur nÃ¤chsten Ã□nderung gelten:

Bei Tastendruck Â□am PromptÂ□ 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 Â□gelistetÂ□ erkannt, erfolgt einfach weitere Bildschirmausgabe bis mit Enter abgeschlossen wird. Dann wird der zwischen aktueller Cursorposition und letztem Â□davor liegendenÂ□ Prompt liegende Text als Kommando, von fÃ¼hrenden und anhÃ¤ngigen Leerzeichen gesÃ¤ubert, in den Befehlspuffer Ã¼bernommen. Also so ein wenig a la Full-Screen-Editor. Ist nun das erste Zeichen im Puffer ein Punkt Â□.Â□, 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 Â□internerÂ□ 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 (Ã¶ffnen 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 zusÃ¤tzlichen Informationen mit abgespeichert wird. Damit wird eine derartige Datei um 10H Bytes lÃ¤nger und ich speichere die Information zur Herkunft der Daten -Anfangsadresse, LÃ¤nge, Speicherbank- und Page (da mein TINY dafÃ¼r 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.

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.

FÃ¼r Datei-Speicheroperationen habe ich 3 Befehle implementiert:

- SD Â interaktives Speichern aus dem externen Datenspeicher,
- SP- analog SD fÃ¼r den aktuellen Programmspeicher
- S_Dateiname.ext Â setzt bei bekannten Erweiterungen, wie bas (BASIC), fth (FORTH) gÃ¼ltige Vorgabewerte fÃ¼r Ablageadresse, Startadresse etc.

Analog existieren auch 3 Ladebefehle.

Bekannte Datei(-typen), die Ã¼ber ein Startadresse verfÃ¼gen und per L_Dateiname.ext geladen werden, werden dann ausgefÃ¼hrt. (das JUTE-Forth in 5 Sekunden geladen und gestartet ist schon ein VergnÃ¼gen!)

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 Â bei mir jetzt Kommando ÂXÂ im 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 MÃ¶glichkeiten arbeite ich nun schon eine Weile mit viel SpaÃ, habe mich inzwischen neuen TINY_Projekten zugewandt und mÃ¶chte den Komfort der schnellen Datenspeicherung nicht mehr missen.

VGA-Anschluss und Pc Â Tastaturanschluss sind in der Zwischenzeit prinzipiell realisiert. Nach Beseitigung der letzten Bugs, werde ich darÃ¼ber berichten.

Hinweise fÃ¼r Verwendung des ROM-ImagesÂ Unterschiede zum Standard-TINY :

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

Das ROM-Image lÃ¤uft definitiv nicht ad hoc in einer Standard-Hardwareumgebung! D.h. es lÃ¤uft schon, nur es sind keine Tastatureingaben mÃ¶glich, da ich zwecks Schaffung eines mÃ¶glichst durchgehenden RAM-Bereiches Alles, was I/O-Operationen anbelangt, in die obersten 2k ÂgepacktÂ habe. In genau diesem ÂSystem-RAMÂ-Bereich liegt also wie gehabt auch der Bildschirmspeicher, dort sind die BIOS-Zellen, die RTC, die Adressen des Vinculum, Pufferspeicher fÃ¼r Flashroutinen

etc... FÃ¼r das Ausprobieren im Emu von Jens MÃ¼ller spielt es keine Rolle, deshalb empfehle ich fÃ¼r das Â„Mal schnell anguckenÂ“ den Emu.

Die Software hat noch alpha-Status, also etliche Bugs, viel internen MÃ¼ll. So die konkrete VDip-Hardware nicht vorhanden ist, wird die Routine abgebrochen. Deshalb lÃ¤uft das auch nicht im Emulator.

In meinem System sind Daten und Programmspeicher getrennt (P34 =/DM). Bei den meisten Menupunkten und Befehlen ist deshalb die EinstellmÃ¶glichkeit fÃ¼r eine Speicherbank und Â„seite vorgesehen. Das bleibt im Emu und im Standard-Tiny also wirkungslos. Das reale System verfÃ¼gt Ã¼ber Â„Bios-MerkzellenÂ“, untergebracht in einer Echtzeituhr. Davon ausgehend werden beim Starten bzw. nach Reset erst etliche Such-und Ã„berprÃ¼fungsvorgÃ¤nge ausgefÃ¼hrt um ggf. ein im Datenspeicher oder im EPROM des Programmspeichers abgelegtes Betriebssystem zu laden. Ich habe, wie schon oben erwÃ¤hnt, durchgehenden RAM-Bereich im Programmspeicher, lade das Betriebssystem aus einem Festwertwertspeicher (EPROM, FLASH, Zeropower-RAM...) in den RAM. Das Starten dauert also lÃ¤nger. Im Startbildschirm erscheint Datum und Uhrzeit. Im Emu hat auch das sonst keine lauffÃ¤higkeitsbedingte 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 fÃ¼r den 8830 gedacht. Altes Â„RÂ“ ist bei mir Â„HÂ“. Der Aufruf der einzelner Befehle aus dem Hauptbildschirm erfolgt durch den zugeordneten Buchstaben, wie gewohnt. Zwischen den Menuseiten wird mit + bzw. Â„ umgeschaltet. Der Start eines Programms erfolgt aus dem Grundmenue heraus mit Â„CxxxxÂ“ Dabei ist xxxx- Adresse in Hex. Damit wird das umstÃ¤ndliche Prog aufrufen, G adr eingeben, L adr eingeben verkÃ¼rzt, ist aber auch mÃ¶glich.

FÃ¼r alle weiteren ErklÃ¤rungen 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**