4k-TINY-Emulator auf Propellerchip (Version V2.0ß)

Kurzbeschreibung:
Auf einem Propellerchip wird der TINY in der 4k-Systemversion z.Z. (Arbeitsstand 01.07.2012) möglichst vollständig so emuliert, dass die Ausführung der vorhandenen Software in Form von BASIC und MC-Programmen erfolgen kann, sowie die eingebauten Menus PROG, DATA, BASIC,SAVE,LOAD auch zum Erstellen eigener Programme nutzbar sind.
Erreicht wird in etwa die Originalgeschwindigkeit des TINY.
Der TINY-ROM, sowie der interne BASIC-ROM des UB8830 sind geringfügig an die realen Hardwarebedigungen angepasst. Insbesondere die SAVE- und LOAD-Routine wurde verändert, da als Speichermedium hier eine SD-Karte, nicht mehr Tonband benutzt wird. Beim Laden wird deshalb der Dateiname als Eingabe abgefragt, ebenso die Ladeadresse, die als Standardwert aber aus der Datei als Vorschlag angezeigt wird.
Als zusätzliche Befehle, jedoch ohne Darstellung als Menueintrag sind integriert:
D-Directory, X-Delete File, C –Call Hexadresse.

Das “Standard-Rom” wird bei Start des Propeller als Grundsystem automatisch initialisiert, steht also immer zur Verfügung. Wird jedoch auf der SD-Karte eine Datei namens „TINY4K.SYS“ bei Start des Emulators gefunden, so wird diese anstelle des „eingebauten“ ROM-Bereiches geladen.

Es wurden weitere Steuertasten(kombinationen) in den Emulator integriert, die den Bedienkomfort erhöhen:

ALT-GR	+ F12	Reboot Propeller
SHIFT	+ F12	Reset TINY, Raminhalt (also ggf geladene Software) bleibt erhalten
	F12	WAIT, bringt den Z8 in einen Wartezustand, bzw. startet Z8
SHIFT	+ F10	Aus-/einschalten Interrupsystem
	F10	Aus-/einschalten IRQ4 des Z8 (im original Bildschirminterrupt)

Bildschirmausgabe erfolgt über VGA. Bei mir findet ein TFT-Flachbildschirm Verwendung, analog Röhren-Monitorausgabe ist ebenfalls mit den mir noch zur Verfügung stehenden Monitoren erfolgreich getestet.

Besonderheit der Registerverwendung von R0, R1 des Z8 in der Emulation:
In der Originalhardware repräsentiert R0 bzw. R1 den Port0/1 des Z8, der im Tiny in der Funktion des Daten-/Adressbusses zur Speicheransteuerung genutzt wird. Bei der Emulation vom ROM bzw. RAM im internen Speicher des Propellers spielen Ein- bzw. Ausgaben über diese Ports bzw. Register keine Rolle. Andererseits kann bei bestehender TINY-Software auch nicht auf diese Register in der originalen Hardware zugegriffen worden sein, eben wegen der o.g. Verwendung der zugehörigen Portleitungen als Busleitungen
Damit ergab sich die Möglichkeit sie als Übergaberegister für Kommandos bzw. Daten vom TINY-System an die Propellerumgebung zu nutzen. Davon wird wie folgt Gebrauch gemacht:

R0 – Kommandoregister, beschreiben des Registers mit einem Wert führt zur Ausführung eines, dem Wert zugeordneten Kommandos, Beschreibung der Kommandos s.u.

R1 – Datenregister, eingeschriebener Wert wird bei Kommandoausführung (ausgelöst durch beschreiben von R0) als Parameter mit übergeben. Bei bestimmten Kommandos wird in diesem Register ein Rückgabewert geliefert (z.B. Fehlercode bei SD-Card-Routinen).

Kommandobeschreibungen

$00 – grafischer TINY-Bildschirmspeicher in den Bitmapspeicher bringen, incl. notwendiger Umrechnungen. Keine Parameterübergabe, kein Rückgabewert. Wird nach jeder anderen Kommandoausgabe wieder als Standard gesetzt.
$10 – SD-Card starten (mounten), kein Parameter, Fehlernummer als Rückgabewert.
$11 – Datei speichern, kein Parameter, Fehlernummer als Rückgabewert.
Dateiname, Start- und Endadresse sind im ehemaligen „Kassetenpufferspeicher“ ($FC00) vorher abzulegen.
$12 – Datei-Header einlesen in Puffer ab $FC00, kein Parameter, Fehlernummer als Rückgabewert, Einzulesender Dateiname wird in Puffer übergeben, am Ende steht im Puffer der Dateiname, sowie – wenn Datei vorhanden – Start- und Endadresse.
$13 – Dateiinhalt laden, kein Parameter, Fehlernummer als Rückgabewert.
Geladen wird an die im Puffer stehende die Start- und Endadresse
$14 – Directory laden, kein Parameter, Fehlernummer als Rückgabewert,
$15 – Datei löschen, kein Parameter, Fehlernummer als Rückgabewert,
Gelöscht wird die Datei, deren Name im Puffer steht.
$1C – Modus der seriellen Schnittstelle einstellen, Modus steht als Parameter in R1
Aufbau des Modusbytes siehe tiny_4k_def.spin
$1D – Baudrate Faktor 1, Faktor wird als Parameter in R1 übergeben
$1E - Baudrate Faktor 2, Faktor wird als Parameter in R1 übergeben und Baudrate berechnet nach der Formel: Fakor1 x Faktor2 x 10, Rückgabe $FF in R1
$1F – SD-Card stoppen und seriellen Treiber starten mit Modus- und Baudratenübergabe Fehlernummer SD-Card ist Rückgabewert in R1
$24 – Bitmapspeicher beschreiben mit $00, kein Parameter, kein Rückgabewert,
$25 – Bitmapspeicher beschreiben mit 4-Byte-Muster, kein Rückgabewert,
Parameter ist die gerade Registernummer des Z8-Registers, ab dem in 4 aufeinanderfolgenden Registern das Muster gespeichert ist,
$26 – Bitmapspeicher negiert darstellen, kein Parameter, kein Rückgabewert,
$27 – Bitmapspeicher auf originale Darstellung schalten, kein Parameter, kein Rückgabewert,
$28 – Bildschirmblinken 1, kein Parameter, kein Rückgabewert,
$29 – Bildschirmblinken 2, kein Parameter, kein Rückgabewert,
Blinkfarben, Dauer und Anzahl ist in der def-Datei festgelegt, bei Blinkende wird zur aktuellen Farbkombination zurückgekehrt, Blinken erfolgt toggeln zwischenBlinkfarbe und Standardfarbe
$2C – Standardfarben entspr. Def-Datei setzen, kein Parameter, kein Rückgabewert
$2D – Hintergrundfarbe übergeben, Parameter ist Farbe, kein Rückgabewert
$2E – Vordergrundfarbe übergeben, Parameter ist Farbe, kein Rückgabewert
$2F – Farbkombination setzen, kein Parameter, kein Rückgabewert

Weitere Kommandos sind im Rahmen des noch zur Verfügung stehenden Speicherplatzes des Prop denkbar und möglich. Die Abfrageroutinen dazu sind zu finden in „calc-z8emu“ (Assembler) bzw. im Hauptobjekt als Spin-Schleife. An diese Stellen wäre dann der entsprechende Programmkode einzubinden.

Besonderheiten der seriellen Schnittstelle gegenüber der Originalhardware

Die SIO im Z8-Original bezieht ihren Takt vom Timer T0 und wird nur dann aktiv, wenn durch entsprechende Programmierung des Registers P3M das Leitungspaar P31/37 als SIO Ein/Ausgang eingestellt wurde. Sie hat praktisch nur einen 1 Byte Lese/Schreibpuffer. Das Schreib-/Lesereister ist Register $F0. Nach erfolgter Ausgabe eines Bytes bzw. Empfang eines Bytes wird erfolgt eine Interruptanmeldung. Es kann ohne bzw. mit (ungerader) Parität mit jeweils gleicher Baudrate für Senden und Empfang gearbeitet werden.
Die Emulation unterscheidet sich in diesen Punkten erheblich. Erstens arbeitet die serielle Ausgabe unabhängig von T0 und damit von der vorherigen Programmierung des Timers. Für die eigentliche serielle Steuerung wird nämlich ein eigener COG des Propeller verwendet, der also vollkommen asynchron gegenüber der restlichen Emulation Daten senden bzw. empfangen kann und so auch sein eigenes Timing zur Baudratenfestlegung erzeugt.. Die Ablage der Daten erfolgt in einem eigenen Puffer der im COG-Ram lokalisiert ist. Die Puffergröße ist im Rahmen des zur Verfügung stehenden COG-Rams in der def-Datei festlegbar. Ebenso werden dort die Standardbaudrate, sowie der Arbeitsmode festgelegt. Diese Werte können zur Laufzeit des Emulators durch entspredchende Kommandos (s. oben) aber geändert werden. Auf die Paritätserkennung wurde verzichtet, dafür jedoch schon erwähnte Arbeitsmodes integriert. So kann mit bzw. ohne Hardwarehandshake (DTR/CTS) gearbeitet werden und die Polarität der Signale (H/L-Aktiv) eingestellt werden. Als Handshakesignale fungieren P24/P25 des Prop.
Es wird stets mit gleicher Baudrate gesendet und empfangen.
Der serielle Eingang bzw. Ausgang ist identisch mit den zur Programmierung des Propeller verwendeten Portleitungen P30,P31.
Unabhängig der Einstellungen in P3M erfolgen immer folgende Abläufe:
Bei Empfang eines Bytes gelangt dieses in den Empfangspuffer des benutzten Cog, und wird, wenn das Z8-Empfangsregister $F0 zuvor bereits gelesen wurde, dorthin weitergereicht. Ein Beschreiben des Registers $F0 führt umgekehrt zur umgehenden Weitergabe an den Sendepuffer des COG.
Wird in P3M SIO-Betrieb vereinbart, so erfolgt zusätzlich zu o.g. Vorgängen entsprechende Anmeldung von Interrupts.

Zusammenfassung der Details der Emulation

- TINY-ROM von 0000h – 27FFh durchgängig
- RAM-Bereich von E000h – FFFFh
- Tastatureingaben von PC-Tastatur werden in „originale“ Tastaturmatrixbelegungen umgesetzt. (z.Z. sind noch nicht alle Tastenbelegungen umgesetzt)
- Soundausgabe/ Timerausgabe über Portpin des Propellerchip (P7)
- Emulation des Port2 des Z8/TINY über P8…P15 des Propeller incl. Handshake
- Original IRQ4-Bildschirmroutine wird nicht benutzt.
- Datenspeichermedium ist SD-Card, Fat-Dateisystem, keine Unterverzeichnisse; 8.3-Dateinamen (Großbuchstaben)
- Serielle Ausgabe, erfolgt über die Portpins, die gleichzeitig zum Programmieren des Propeller benutzt werden.
  Die Z8-SIO wird dabei nicht emuliert, es wird auf einen separaten Treiber zurückgegriffen, der einen E/A-Puffer besitzt. Standardeinstellungen in der def-Datei. Keine Paritätserkennung!
- Interruptsystem ist implementiert, jedoch ohne Prioritätssteuerung (Register IPR).

Die einzelnen Cog’s haben folgende Aufgaben:

Cog 0 : „Startcog“, startet alle anderen Funktions-Cogs, dient dann auch zur Kommunikation(Befehlsaustauch) der Cogs untereinander.
Startet bei Bedarf SD-Cog zwecks SD-Routinen

Cog 1 : „VGA-Cog“, Treiber für VGA-Ausgabe, gibt Bitmap-Speicher auf Monitor aus.

Cog 2 : „Berechnung-Cog“, liest emulierten TINY-Bildwiederholspeicher, rechnet ihn nach Bitmap-Darstellung um. Kann auch für weitere Bildschirmorientierte Aktionen (Farben etc.) über spezielle Kommandos benutzt werden

Cog 3 : „Keyb-Cog“, Treiber für PS2-Tastatur und
Emulation Tastendruck der TINY-Tastaturmatrix., sowie Steuertastenfunktionen.

Cog 4 : „RS232-Cog“, Treiber für serielle Ausgabe, Zeichenpuffer im Cog-Ram.

Cog 5 : „INT-I/O-Cog“, steuert die Emulation der physischen Ansteuerung der Portpins des Propellers für P2-Emulation, serielle Schnittstelle etc. Interruptsystem

Cog 6 : „Timer-Cog“, emuliert T0 und T1 des Z8

Cog 7 : „Z8-Core-Cog“, emuliert, z.T. unter Verwendung von LMM, d.h. Nutzung des HubRam als Programmspeicher den Z8 und dient der Ansteuerung des virtuellen ROM/RAM-Bereiches des TINY

Cog 8 : „SD-Cog“, wird bei Bedarf gestartet, wenn SD-Card-Kommunikation stattfinden soll.

Der komplette Registersatz des Z8 (00..FFh) liegt ebenso im Hub-Ram des Propellers, wie der emulierte ROM-/RAM-Bereich und die virtuellen Tastaturmatrixzellen.

Dateiformat der SD-Kartenspeicherung
Abspeicherung und Laden von SD-Card erfolgt mittels SAVE/LOAD menugesteuert.
Das Dateiformat entspricht dem „*.jtc“ –Formats des PC-Emulators von J. Müller.
und besteht aus einem 80h-Byte großen Vorblock, der mit dem Dateinamen beginnt. Hier ist auch momentan der Unterschied zum jtc-Original:
Dort muss der Name stehen, unter dem das Programm gespeichert ist und zwar in der Form 8.3 und in Großbuchstaben. Bei Beachtung dieser Regel ist somit mittels Hexeditor auf dem PC eine jtc-Datei problemlos anpassbar.

Hardwarevoraussetzungen seitens Parallax-Propellerhardware
Es wird nur eine Minimal-Hardware-Umgebung für den Propellerchip benutzt, d.h. ein (selbstgebautes) Parallax-Demoboard. Dies ist ergänzt um einen SD-Cardanschluss zur Datenspeicherung/Laden von Programmen. Benutzt werden dazu die Portpins 0-3.

Anpassung an andere Anschlussbelegungen (Hydra etc.) ist möglich und erfolgt durch ersetzen der entsprechenden Einträge in der Datei „**tiny_4k_def.spin**“

Installation
Die zip-Datei wird in ein Verzeichnis auf dem PC entpackt, im Ordner SDCard sind einige Beispielprogramme, die auf SD kopiert werden können. Nach Start der Propeller- IDE, ich verwende das Original von Parallax, nicht BST(geht aber genauso), wird die Datei „tiny_4k_beta20.spin“ aufgerufen, gegebenenfalls in „tiny_4k_def.spin“ die für die individuelle Hardwarebasis notwendigen Anpassungen vorgenommen, und dann wie üblich in den EEPROM des Propellersystems geflasht.

Uwe Nickel
u.nickel@lycos.com