

**KLEINCOMPUTER**



**KC85**

**M041**

**2 x 16 KB EEPROM**

**Nachbau des Moduls M040  
für den KC85/2-5  
als Neuentwicklung  
M041 - 2 x 16 KB EEPROM**



Revision 3.0 / Korrigierte Ausführung

# KLEINCOMPUTER

---

## KC 85

Beschreibung zu M041 2 x 16 KB EEPROM



veb mikroelektronik  
'wilhelm pieck'  
mühlhausen

## KC-CLUB

Mario Leubner / René Nitzsche

Herausgeber: KC-Club

Autoren: Mario Leubner, René Nitzsche

Gestaltung und Layout: René Nitzsche

Fotos und Grafiken: René Nitzsche

Der Vertrieb dieser Druckschrift erfolgt ausschließlich durch den Herausgeber.

Ohne Genehmigung des Herausgebers ist es nicht gestattet, das Buch oder Teile daraus nachzudrucken oder zu vervielfältigen.

Hinweise, die zur Verbesserung dieser Dokumentation führen, werden gern entgegengenommen.

Redaktionsschluß: Mai 2017

SRN 2017 ©

## **Inhaltsverzeichnis**

1	Vorwort.....	3
2	Einleitung .....	6
3	Modulhandhabung .....	7
3.1	Stecken und Entfernen des Moduls .....	7
3.2	Modulkennung und Zuweisung .....	8
3.2.1	Modulkennung.....	8
3.2.2	Modulzuweisung .....	9
3.2.2.1	Betriebszustände.....	9
3.2.2.2	Speicheradressierung .....	10
3.2.3	Modulselektion .....	12
3.2.4	Steuerbytedefinition .....	13
4	Technische Beschreibung .....	14
4.1	Systemkonfiguration .....	15
4.2	ROM-Adressierung.....	15
5	Entwicklung, Aufbau und Inbetriebnahme .....	16
5.1	Modulentwicklung.....	16
5.2	Aufbauhinweise .....	17
5.3	Inbetriebnahme und Benutzung des M041 .....	21
6	Anwendungsfall / Programmerstellung .....	26
7	Anlage A – Schalt- und Belegungspläne .....	29
7.1	Belegungsplan – M041.....	29
7.2	Schaltplan – M041 / Submodul 0 .....	30
7.3	Schaltplan – M041 / Submodul 1 .....	31
7.4	Schaltplan – M040.....	32
8	Anlage B – Bestückungsliste .....	33
9	Anlage C – Quellcode SRNSHOW .....	35

## **1 Vorwort**

Es ist nicht üblich, daß in den Handbüchern zu den Erweiterungsmodulen für den KC85/2-5 ein Vorwort steht und dazu noch ein Persönliches. An dieser Stelle möchte ich mit dieser Unüblichkeit brechen. Es wird auch das einzige Kapitel sein, welches in der Ich-Form geschrieben ist.

Am Anfang der neuentdeckten Leidenschaft für den KC85 - welche so in etwa im Jahr 2009 begann - stand das Sammeln jedweder nicht vorhandener Hardware rund um den KC85. Neben den Aufsatzgeräten D002, D004, Floppy-Disk, der Komfortastatur D005, diversen Peripheriegeräten - Drucker, Plotter, Erika 6007 - wurden auch zahlreiche Module erworben. Darunter ganz Klassische, von MPM Mühlhausen entwickelte, aber auch Neuentwicklungen von begabten Elektronikern und Programmierern.

Bei weitem nicht jedes Modul konnte in die Sammlung integriert werden, da es partout nicht käuflich vorhanden war. Eines dieser Module ist das M040. Warum war auch dieses Modul für mich so interessant?

Ab und an rafft es mich und dann programmiere ich ein wenig. Hauptsächlich handelt es sich um Grafikanwendungen – Bildschirm-schoner, Slide-Shows, kleine Spiele. Meine Lieblingssprachen sind C++ und PERL. Das Betriebssystem ist mir dabei recht egal. Ob Linux, Windows, Unix, OpenVMS oder auch ganz andere wie CAOS oder TOS; es spielt keine Rolle. Viel wichtiger ist das Vorhandensein einer ausgereiften Entwicklungsumgebung mit integrierten Werkzeugen, u.a. Debugger, Linker und Compiler. Eine umfangreiche Funktionsbibliothek ist ebenso wichtig.

Zu den leider vergessenen Sprachen zählen Assembler und Pascal.

Vor geraumer Zeit hatte ich für den KC85 ein kleines CAOS-Programm entwickelt. Es handelt sich um eine Art Diashow. Dargestellt werden Bilder von Ausflügen und Urlauben in ganz Deutschland.

Weitere und ausführlichere Informationen zu diesem Programm gibt es im Kapitel "6 Anwendungsfall / Programmerstellung".

Das M040 bietet die Möglichkeit so konfiguriert zu werden, daß es als Autostartmodul zum Einsatz kommen kann. Und genau das war meine Motivation; mein kleines Programm so umzuschreiben, daß es autostartfähig wird und sofort beim Einschalten des KC startet. Dazu benötige ich ein M040 - und weitere Voraussetzungen, welche jedoch als schon erfüllt betrachtet werden konnten.

Mehr sollte es gar nicht werden. Mir hätte es gereicht. Daß letztendlich etwas viel Besseres entstanden ist, erfreut mich sehr.

Ich habe es mir zu einer kleinen "Tradition" gemacht, für "meine" Platinen eigene Gehäuse zu entwerfen und per 3D-Druck anfertigen zu lassen. So auch dieses Mal. Dank der schicken und personalisierten Blende von Maik Trompeter kommt dabei auch noch etwas für mein Auge heraus.



**Abbildung 1: Modulgehäuse und Blende für das M041**

Für meine Anforderungen und Zwecke hätte ein M040 vollkommen ausgereicht. Die Schaltpläne waren vorhanden - wunderbar sauber von Mario Leubner neugezeichnet. Und es hätte sogar eine kleine Erweiterung stattgefunden, nämlich die ebenfalls von Mario Leubner entwickelte Modifizierung, die den Einsatz von EEPROM- statt EPROM-Schaltkreisen ermöglicht. Damit entfällt das lästige Herausnehmen der EPROM, löschen dieser mit anschließender Neuprogrammierung und dann wieder Einsetzen in die Fassungen.

Warum ist es nicht dabei geblieben? Manchmal frage ich mich das auch. Meine Kenntnisse zur Entwicklung neuer Schaltungen und Funktionen sind äußerst bescheiden, um nicht zu sagen, gar nicht vorhanden. Das Nachbauen von funktionsfähigen Schaltungen bereitet mir Spaß. Häufig kann ich dann beim Re-Layout der Leiterplatte dieses Layout optimieren. Die dazu notwendigen Kenntnisse habe ich mir im Laufe der letzten Jahre nach und nach angeeignet. Nur warum sollte ich das im stillen Kämmerlein machen? Es gibt andere Menschen, die interessieren sich vielleicht auch für einen puren Nachbau.

Mit Mario Leubner habe ich schon einmal zusammengearbeitet. Damals ging es um die Entwicklung des M030. Zusammen mit Wolfgang Harwardt und Mario Leubner wurde ein leistungsstarkes Modul entwickelt.

Zunächst war von mir beim Nachbau des M040 gar keine Zusammenarbeit mit Mario Leubner geplant. Nicht, weil ich nicht wollte, sondern weil ich keinen Grund dafür sah. Mein Wunsch war es jedoch, auf Marios Hilfe zurückgreifen zu dürfen, wenn ich Verständnisprobleme beim Nachbau habe und Mario gewährte mir die Hilfe auch.

Mario war es auch, der durch seine Gedankenanstöße dafür sorgte, daß es zur Entwicklung des M041 kam. Er setzte mir keine fertigen Lösungen vor, sondern zwang mich zum Denken. Kam es dann doch manchmal zum Stillstand, dann half er mit Schaltungsvorschlägen. Natürlich ist letzten Endes ein Großteil der Entwicklung des M041 von Mario Leubner vorgenommen worden. Aber mit ein klein wenig Stolz kann ich sagen, daß auch von mir Teile der Schaltung entworfen wurden.

Wie schon seinerzeit beim M030 habe ich auch dieses Mal sehr viel dazugelernt und zum ersten Mal habe ich eine Schaltung in ihren Grundzügen auch verstanden und war in der Lage, selbst Teilschaltungen zu entwerfen.

Die Zusammenarbeit mit Mario ist immer sachlich und freundlich gewesen. Er hat eine seltene Gabe und manchmal wünschte ich mir, er würde auch einmal etwas mehr Emotionen zeigen ☺. Es hat immer Spaß gemacht und war für mich wieder eine Fundgrube des Wissens.



## **2 Einleitung**

Das Modul M041 2 x 16 KB EEPROM basiert auf dem Modul M040 und erweitert die Speicherkapazität des KC85/2 und seiner Nachfolgetypen um 2 x 16 KByte Festwertspeicher (EEPROM). Dabei beinhaltet das M041 zwei Submodule, wobei jedes Submodul 2 x 8 KByte Festwertspeicher besitzt.

Das Modul M041 kann mit bis zu 4 EEPROM-IC bestückt werden, wobei empfohlen wird, ein Submodul jeweils vollständig zu bestücken.

Mit dem Erwerb des EEPROM-Moduls erschließen sich neue Anwendungsbereiche durch Ablegen von Festprogrammen, wie z.B. Betriebssystem, Testmonitor, Assembler, Interpreter oder Compiler und eigene Anwenderprogramme.

Nach dem Einschalten und der entsprechenden Modulzuweisung sind die Programme abarbeitungsfähig.

Das Hauptmerkmal des M041 ist die Möglichkeit, den Festwertspeicher "in system" programmieren zu können. Hinzukommt die Autostartfunktionalität, wenn das Modul im Schacht 8 des Grundgerätes steckt und das Kennbyte entsprechend auf "01" konfiguriert wurde. Dazu existiert die Möglichkeit per DIP-Schalter, welcher ohne Öffnen des Moduls bedient werden kann.

Die beiden 8 KByte-Blöcke eines jeden Submoduls können rotiert werden, so daß entweder EEPROM 0 oder EEPROM 1 auf, zum Beispiel, der Adresse C000 eingeblendet wird, wobei der jeweils andere dann verdeckt auf Adresse E000 liegt. Damit kann auf dem M041 auch das Programm TYPESTAR abgelegt werden.

Zur Bedienung des Moduls wird auf das Kapitel "5.3 Inbetriebnahme und Benutzung des M041" verwiesen. Das Kapitel "4 Technische Beschreibung" beschreibt die technischen Details des M041.

### 3 Modulhandhabung

#### 3.1 Stecken und Entfernen des Moduls

Das M041 2 x 16 KB EEPROM kann prinzipiell in jedem Modulsteckplatz betrieben werden. Jedoch ist dabei die Modulpriorität in der gewählten Systemkonfiguration zu berücksichtigen. Es sind weiterhin die nachfolgenden Informationen und die unter Kapitel "4.1 Systemkonfiguration" aufgeführten Hinweise zu beachten.

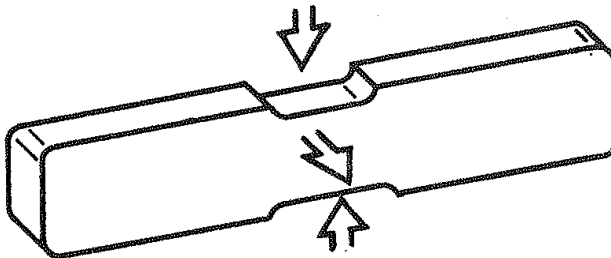
Die Modulprioritätskette muß immer geschlossen bleiben. Es sind also erst im Grundgerät der Steckplatz 8 (rechts), danach der Steckplatz C (links) und anschließend weitere Steckplätze von Erweiterungsaufsätzen in der vorgegebenen Reihenfolge zu belegen.

**Achtung !**

Das Stecken bzw. Entfernen des Moduls darf nur bei ausgeschaltetem Computer bzw. Aufsatz erfolgen!

Das Modul ist durch folgende Handgriffe zu stecken.

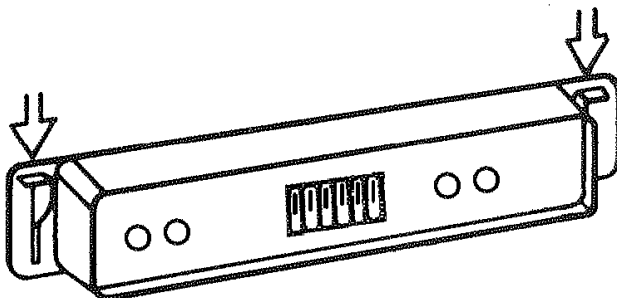
- a) Den Computer bzw. Aufsatz ausschalten.
- b) Die Kappe des Modulschachtes ist durch leichten Druck von Daumen und Zeigefinger auf die Griffflächen abzunehmen.



- c) Das Modul bis zum fühlbaren Einrasten einschieben (hervorstehender Rand des Moduls liegt unmittelbar an der Gerätestand an).
- d) Nun kann der Computer bzw. Aufsatz eingeschaltet werden.

Zum Entfernen des Moduls aus dem System sind folgende Schritte notwendig.

- a) Den Computer bzw. Aufsatz ausschalten.
- b) Den linken und rechten Zeigefinger unter den Modulkopf legen und mit den Daumen die seitlich am Modul befindlichen Hebel gleichzeitig nach unten drücken. Dabei rastet das Modul aus und wird etwa einen Zentimeter aus dem Gerät herausgeschoben. Nun das Modul aus dem Schacht nehmen.



- c) Die Kappe auf die Schachthöffnung stecken.

## **3.2 Modulkennung und Zuweisung**

### **3.2.1 Modulkennung**

Jedes Modul erhält eine für es charakteristische Modulkennung, die durch das Kennbyte gekennzeichnet ist. Dieses Kennbyte widerspiegelt den Modultyp bzw. die innere Struktur des Moduls. Vom Nutzer kann hierauf softwaremäßig kein Einfluß genommen werden, da es sich um funktionsbedingte Festlegungen des Herstellers handelt. Das Kennbyte kann durch den Prozessor auch im inaktiven (nicht eingeschalteten) Zustand des Moduls gelesen werden. Dadurch kann sich der Nutzer jederzeit in einem ausgebauten System einen Überblick über die verfügbaren Module verschaffen und in Abhängigkeit davon seine Entscheidung treffen.

Das M041 kann 4 verschiedene Kennbytes annehmen. Dies sind die Kennbytes 01 (Start-ROM), F1, F8 und FC. Die Möglichkeit zum Ändern des Kennbytes bietet ein von außen zugänglicher DIP-Schalter.

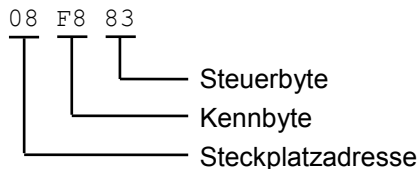
Das Kennbyte kann über den Befehl SWITCH und die Angabe der Modulsteckplatzadresse bzw. über das entsprechende Betriebssystem-Unterprogramm (vgl. Systemunterlagen KC 85) gelesen werden, was folgendes Beispiel zeigt.

Beispiel:

Das Modul steckt im Schacht 8, das Kennbyte soll gelesen werden. Es ist einzugeben:

```
SWITCH 8
```

Nach Drücken der ENTER-Taste erscheint folgende Information auf dem Bildschirm.



Die Steckplatzadresse 08 gibt an, daß sich das Modul im Grundgerät (0) befindet und Steckplatz-Nr. 8 hat.

Das Kennbyte F8 zeigt an, daß das M041 als 16 KByte ROM-Modul konfiguriert ist.

Das Steuerbyte 83 signalisiert, daß das Modul ab der Speicheradresse 8000H eingeblendet ist und beschrieben werden kann.

## **3.2.2 Modulzuweisung**

### **3.2.2.1 Betriebszustände**

Es werden zwei Betriebszustände des Moduls unterschieden.

#### **1. INAKTIV**

Die "Modul-EIN"-Diode leuchtet nicht. Das Modul ist vom Prozessor getrennt.

#### **2. AKTIV**

Die "Modul-EIN"-Diode leuchtet. Das Modul kann entweder gelesen oder beschrieben werden.

Der gewünschte Betriebszustand wird im CAOS-Menü über den Befehl

SWITCH mm kk

eingestellt.

Die beiden Parameter dieses Befehls beschreiben folgendes.

- mm Mitteilung an das System, in welchem Modulschacht das zuzuweisende Modul gesteckt ist. Dabei ist die erste Stelle von mm die Nummer des Aufsatzes (im Grundgerät ist diese Stelle Null und kann weggelassen werden). Die zweite Stelle von mm ist die Steckplatzadresse. Im Grundgerät gibt es nur die Steckplatzadressen 8 (rechter Schacht) und C (linker Schacht).
- kk Steuerbyte für das zuzuweisende Modul. Das Steuerbyte enthält die Basisadreßzuordnung für das Speicher-Segment und den Betriebszustand des Moduls. Es enthält also die Steuerbedingungen für das Modul, die vom Nutzer verändert werden können.

Beispiel:

Das Modul soll im Schacht 8 steckend ab der Adresse 4000H schreibgeschützt aktiviert werden. Es ist einzugeben:

SWITCH 8 41

Um auf das Modul zugreifen zu können, ist der RAM 4 abzuschalten.

### **3.2.2.2 Speicheradressierung**

Beide Submodule des M041 können auf die Speicherbereiche mit folgenden Basisadressen zugewiesen werden.

0000H, 4000H, 8000H, C000H

Als Steuerbyte kk ergibt sich daraus:

01/03, 41/43, 81/83, C1/C3

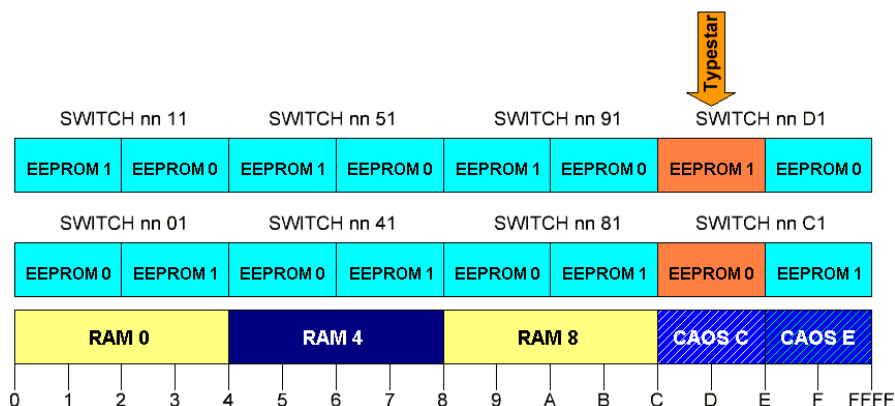
Sollen die beiden EEPROM eines jeden Submoduls im Speicherbereich ihre Zuordnung tauschen (EEPROM 1 liegt vor EEPROM 0), dann ist das Steuerbyte um 10H zu inkrementieren, d.h. zum Beispiel statt C1 wird D1 angegeben.

Es können beide Submodule gleichzeitig aktiv sein, jedoch nicht auf derselben Basisadresse. In der nachfolgenden Abbildung wird gezeigt, wie sich die Speicherzuordnung beider aktiver Submodule darstellt. Dazu werden die jeweiligen EEPROM mit ihren Schaltkreisnamen benannt.

Für die Zuweisung des M041 bzw. beider Submodule ist zu beachten, daß je nach verwendetem Grundgerät die meisten Speicherbereiche bereits standardmäßig durch Speicher belegt sind. Der für das M041 gewünschte Speicherbereich muß gegebenenfalls erst freigeschaltet werden.

Die folgende Abbildung zeigt schematisch die Speichersegmente eines KC85/4 und beider Submodule des M041 (4000H-7FFFH und C000H-FFFFH) dargestellt.

## M041 - RAM-Segmente



**Abbildung 2: Beispielhafte Speicherbelegung beider Submodule des M041**

Wird das M041 ab der Adresse C000H eingeblendet, ist darauf zu achten, daß ein Zugriff auf den zweiten 8-KB-Block nur durch Rotation der beiden 8-KB-Blöcke erfolgen kann, d.h. die beiden EEPROM-Schaltkreise tauschen ihre Adreßzuordnung.

### 3.2.3 Modulselektion

Die bei der Zuweisung des M041 verwendeten Parameter mm (Steckplatzadresse) und kk (Steuerbyte) sind zweistellige Hexadezimalzahlen, die aus je 8 Bit bestehen. Die einzelnen Bits verschlüsseln binär folgende Informationen.

Modulsteckplatzadresse mm

G3 G2 G1 G0 S1 S0 X1 X0

Blockauswahl

Modul	X1	X0
M041 / Submodul 0	0	0
M041 / Submodul 1	0	1

Steckplatz im Gerät

	S1	S0	S1	S0
oben	1	1	1	0
unten	0	1	0	0
	links		rechts	

Gerätenummer

Gerät	G3	G2	G1	G0
Grundgerät	0	0	0	0

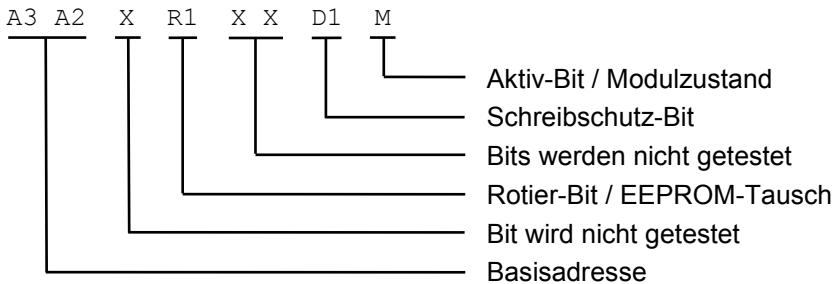
Die folgende Tabelle beschreibt eine Beispielselektion.

Modulsteckplatzadresse								Seite	
binär								hex	
G3	G2	G1	G0	S1	S0	X1	X0		
0	0	0	1	0	1	0	1	15	links

Tabelle 1: Submodul 1 des M041 im Steckplatz 14H und Block 1

### 3.2.4 Steuerbytedefinition

Im Steuerbyte kk werden folgende Informationen binär verschlüsselt.



Aus der vorher dargestellten Zuordnung der Bits ergeben sich für das Steuerbyte die einzusetzenden Werte. In der folgenden Tabelle sind sinnvolle Steuerbytes zur Veranschaulichung aufgeführt.

Steuerbyte											
binär								hex	dez		
A3	A2	X	R1	X	X	D1	M			Basis	Zustand
0	0	0	0	0	0	0	0	00	0	0000H	inaktiv
0	0	0	0	0	0	0	1	01	1	0000H	aktiv / r.o.
0	1	0	0	0	0	1	1	43	67	4000H	aktiv / r.w.
1	0	0	0	0	0	0	1	81	129	8000H	aktiv / r.o.
1	1	0	1	0	0	1	1	D3	211	C000H	aktiv / r.w.

Tabelle 2: Mögliche Steuerbytes des M041



## **4 Technische Beschreibung**

Auf der Platine des M041 finden zwei gleichwertige Module Platz. Jedes Submodul bietet 16 KB ROM-Speicher. Dabei können neben den vorgesehenen EERPOM auch EPROM eingesetzt werden, dann mit der Einschränkung, daß diese nicht "in system" programmiert werden können. Die jeweils 16 KB teilen sich auf zwei 8 KB-EEPROM auf. Gleich dem M033 (Typestar) können diese beiden Blöcke rotiert werden, so daß entweder EEPROM 0 oder EEPROM 1 auf, zum Beispiel, der Adresse C000H eingeblendet wird, wobei der jeweils andere dann verdeckt auf Adresse E000H liegt.

Das Modul bietet einen dreifachen Schreibschutz, wenn EEPROM mit SDP (Software data protection) eingesetzt werden. Als erstes eben SDP, dann zum zweiten per Steuerbyte und zum dritten per Hardware. Dazu befindet sich an der Platinenvorderseite ein DIP-Schalter, mit dem jedes Submodul per Schalter schreibgeschützt werden kann. Dieser DIP-Schalter dient auch zur Einstellung des Kennbytes. Dadurch lassen sich die Kennbytes 01 (Start-ROM), F1, F8 und FC einstellen.

Hauptmerkmale des M041 sind sicherlich die insgesamt 32 KB ROM-Speicher, die Autostartfunktionalität und die Möglichkeit des "in system programmings".

Optisch hat die Front des M041 nichts mehr mit seinem Vorbild, dem M040, gemein. Neben dem bereits erwähnten DIP-Schalter besitzt das Modul 4 RGB-LED in zwei Zweiergruppen, wobei jeweils die linke LED das Submodul 0 repräsentiert.

Die beiden linken LED haben folgende Funktionen.

grün	Das Modul ist aktiv.
blau	Der EEPROM 0 des M041 wird beschrieben.
rot	Der EEPROM 1 des M041 wird beschrieben

Die beiden rechten LED haben folgende Funktionen.

blau	Der EEPROM 0 des M041 ist aktiv (lesen oder schreiben).
rot	Der EEPROM 1 des M041 ist aktiv (lesen oder schreiben).

Hier ist jedoch anzumerken, daß es sich um die Detektierung des /CS-Signals handelt. Diese Detektierung ist vergleichsweise kurz, so daß manchmal nur ein "Glimmen" wahrgenommen werden kann, weil die LED gar nicht so lange angesteuert wird, um ihre volle Leuchtkraft zu entfalten.

Wird das Kennbyte des M041 auf 01 geändert und steckt das Modul im Modulschacht 8 des KC85-Grundgerätes, wird beim Systemstart nach der

Initialisierung des M041 mit der Basisadresse 4000H aktiviert. Danach erfolgt ein Sprung auf die Adresse 4000H, also auf das erste Byte im Modul M041. Mit der Änderung des Kennbytes auf 01 kann also ein beim Kaltstart (POWER ON) oder Warmstart (RESET) selbststartendes Programm erzeugt werden.

## **4.1 Systemkonfiguration**

Das M041 2 x 16 KB EEPROM ist in das Erweiterungskonzept des KC85/2 und seiner Nachfolgetypen voll eingebunden. Die Adressierung erfolgt in 16 KByte-Schritten. Es können auch mehrere M041 im System vorhanden sein, die auf unterschiedlichen Adreßbereichen gleichzeitig oder auf gleichen Adreßbereichen nacheinander genutzt werden können. Bei Nutzung in einem Adreßbereich ist folgendes zu beachten.

Alle Module des Kleincomputers verfügen über eine Modulprioritätssteuerung, d.h. die Priorität fällt mit steigender Modulschachtnummer. Die Module sind in einer Kette aneinandergereiht, die durch die Signale MEI und MEO gebildet wird. Damit ist es möglich, alle in einem Adreßbereich aktiv geschalteten gleichartigen Module in ihrer Priorität so zu steuern, daß nur das jeweils höchstpriorisierte Modul für den Prozessor verfügbar ist. Alle anderen aktiven Module bleiben für den Prozessorzugriff gesperrt. Soll auf ein niedriger priorisiertes Modul zugegriffen werden, müssen alle höher priorisierten Module des gleichen Adreßbereichs inaktiv geschaltet sein. Nach Betätigen der RESET-Taste am KC bleibt das zugewiesene Steuerbyte im Modul erhalten, so daß eine Neuuzuweisung nicht erforderlich ist.

## **4.2 ROM-Adressierung**

Nach der Zuweisung des Moduls durch das Kommando SWITCH mm kk (wobei durch das Steuerbyte die Basisadresse/Anfangsadresse für das Modul festgelegt wurde) gelten für die EEPROM folgende Adreßbereiche (BA = Basisadresse). Beispielhaft wird das M041 folgendermaßen aktiviert. SWITCH 8 41 und SWITCH 9 C1

Submodul 0 / EEPROM 0	BA	4000H - 5FFFFH
Submodul 0 / EEPROM 0	BA + 2000H	6000H - 7FFFFH
Submodul 1 / EEPROM 0	BA	C000H - DFFFFH
Submodul 1 / EEPROM 1	BA + 2000H	E000H - FFFFFH

Nach der Betätigung der RESET-Taste des Computers bleiben das eingeschriebene Steuerbyte im Modul und die Steuerbytetabelle im Computer erhalten. Beim KC85/3 wird aber der BASIC-ROM zugeschaltet.

## **5 Entwicklung, Aufbau und Inbetriebnahme**

Das Modul M041 2 x 16 KB EEPROM wurde im Zeitraum Oktober/2016 bis April/2017 entwickelt. Während dieser Zeit flossen neue Gedanken und Ideen mit ein, so daß aus der ursprünglichen Intention, das M040 als EEPROM-Variante nachzubauen, eine Neuentwicklung wurde.

Da das M041 zwei gleichwertige Submodule beinhaltet, ist das Layout gegenüber dem Vorbild M040 grundlegend verändert worden.

### **5.1 Modulentwicklung**

Wie es bei der Entwicklung eines Produktes üblich ist, wurde nicht auf "gut Glück" ein Schaltplan entworfen und ein Layout gezeichnet. Nach dem der Schaltplan einen Entwicklungsstand erreichte, welcher die Anfertigung eines Prototypen rechtfertigte, wurde ein Layout gezeichnet und der Prototyp hergestellt.

Während des Aufbaus und der Erprobung des Prototypen ergaben sich sowohl Änderungen am Schaltplan, als auch am Layout.

Die im Prototyp nicht vorhandenen und nachträglich eingebrachten LED zur Aktivitätsanzeige zeigten praktisch sofort einen unerwarteten Nutzen. Es war zu erkennen, daß eine ständige Selektierung des zweiten EEPROM eines Submoduls stattfand. Dies ist nicht korrekt. Als Ursache wurde die fehlende /MREQ-MEI-Verzögerung vermutet, welche zunächst nicht vorgesehen war. Dieser Schaltungsteil ist auch bei anderen, ähnlich funktionierenden Modulen nicht vorhanden, so daß davon ausgegangen wurde, daß er nicht benötigt wird. Nachdem dieser nachträglich implementiert wurde, ergab sich ein sauberes "chip select". Dies hatte zur Folge, daß ein weiterer Schaltkreis auf der Platine untergebracht werden mußte. Im Zuge dessen mußte auch der Hardware-Schreibschutz anders gestaltet werden, da das bislang dazu verwendete AND-Gatter für die o.g. Verzögerung eingesetzt wird. Für diesen Zweck wurde ein sehr ausgeklügeltes diskretes AND entworfen.

Der Prototyp sieht vor, die LED zur Programmieranzeige (Schreibzugriff auf den EEPROM) direkt am PIN 1 (/BSY) der EEPROM anzuschließen. Leider gibt keines der Datenblätter der EEPROM Auskunft darüber, wie hoch der Maximalstrom ist, der dem Ausgang bei LOW entnommen werden kann. Um jedes Risiko zu vermeiden, sind die LED nun über eine Emitter-Folge-Schaltung am PIN 1 angeschlossen. Der damit dem Ausgang entnommene Strom liegt im  $\mu\text{A}$ -Bereich. Damit mußten weitere 12 diskrete Bauteile (Transistoren, Widerstände) untergebracht werden.

Die Tests umfaßten Schreibversuche verschiedener Art; sowohl im Byte- als auch im Blockmodus, mit aktiviertem Schreibschutz (Hardware, SDP, Steuerbyte). Für die Leseversuche kam zunächst das Programm TYPESTAR zum Einsatz, da es gleich zwei Merkmale des M041 nutzt: Die Rotierbarkeit der beiden 8 KB-Blöcke und die Autostartfunktionalität. Später wurde das Programm SRNSHOW verwendet (siehe dazu das Kapitel "9 Anlage C – Quellcode SRNSHOW").

Sämtliche Tests und Versuche endeten mit dem erwarteten Ergebnis.

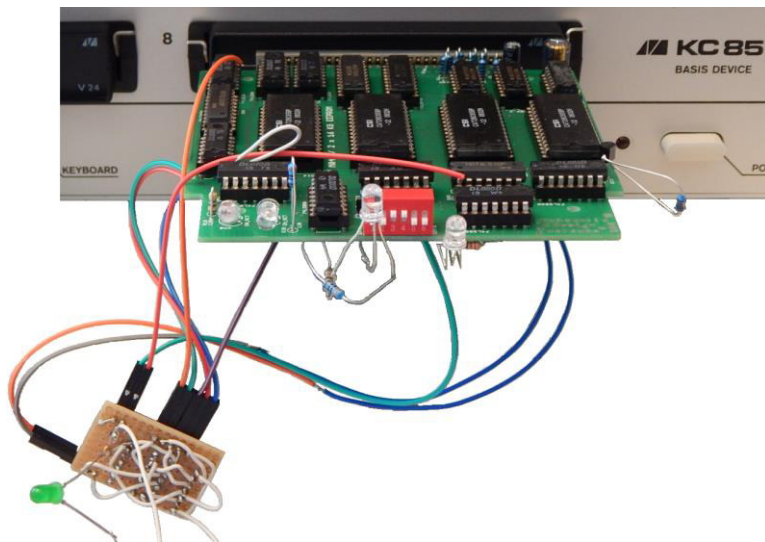


Abbildung 3: Prototypaufbau des M041

## 5.2 Aufbauhinweise

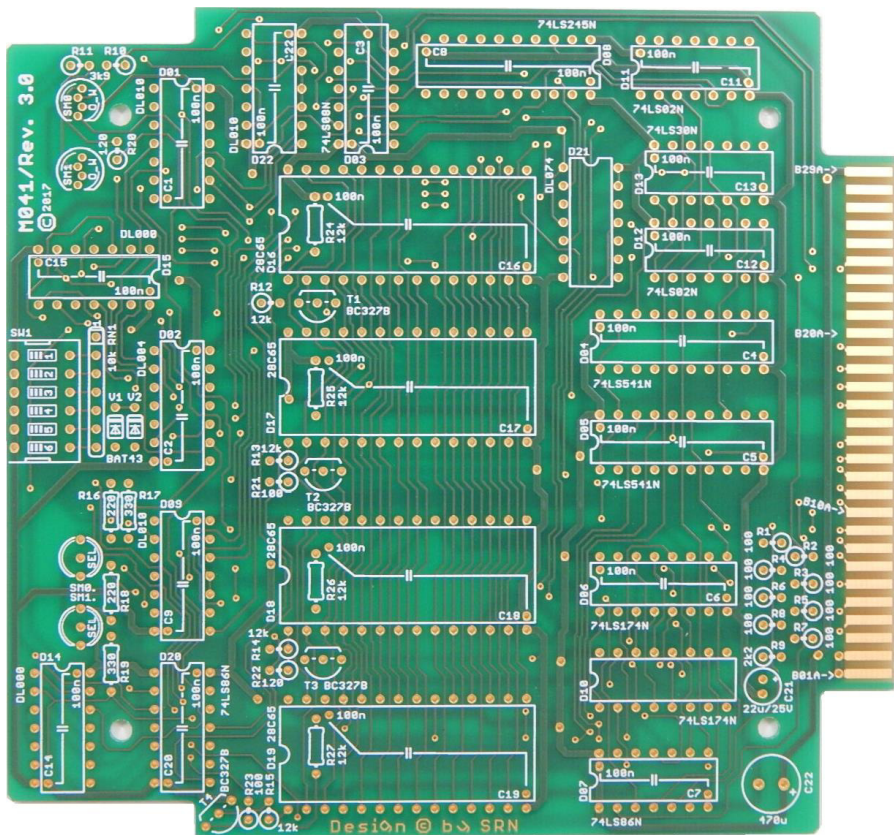
Beim Aufbau ist nicht sehr viel zu beachten. Die Platine sollte leicht zu löten sein. Nachfolgend wird beschrieben, auf was beim Aufbau ggf. geachtet werden sollte.

Einige Bauteile sind unterhalb der Schaltkreisfassungen einzubauen. Dies betrifft alle Abblockkondensatoren (sofern nicht Fassungen mit integrierten Kondensatoren verwendet werden) und die 4 "pull-up"-Widerstände für die /BSY-Ausgänge der EEPROM. Diese Bauteile sollten zuerst bestückt werden. Bei den Kondensatoren ist darauf zu achten, daß diese nur dann mittig liegen dürfen, wenn die IC-Fassung keinen Mittelsteg besitzt. Zwei der 21 Schaltkreise (D10, D21) besitzen keinen separaten Abblockkondensator und "teilen" sich diesen mit den benachbarten

Schaltkreisen. Zum Bausatz gehört eine 14-polige IC-Fassung mit Mittelsteg. Diese ist für D21 vorzusehen.

Zwei der Schaltkreise (D08, D15) sind asymmetrisch eingesetzt. Beim Einsetzen in die Fassung ist dies unbedingt zu beachten, um eine Zerstörung des Schaltkreises zu vermeiden.

Der DIP-Schalter ist soweit wie möglich nach vorn zu "ziehen", damit er später aus der Blende herausragt und die kleinen Hebelchen einfach bedient werden können.



#### Abbildung 4: Unbestückte Serienplatine

Knackpunkt dürften die beiden linken RGB-LED sein. Diese sind 4-polig. Die Pads sind sehr eng beieinander liegend. Hier muß unbedingt mit einer sehr feinen Lötspitze gelötet werden, damit keine Kurzschlüsse entstehen. Ideal ist auch ein sehr feiner Lötdraht.

Dann Beinchen für Beinchen anlöten, dabei nach jedem Anlöten eines Beinchens dieses kürzen und dann das Nächste anlöten.

Da sich herstellungsbedingt der Leuchtpunkt für "Grün" nicht immer an der gleichen Stelle befindet, sollten vor dem Einbauen zwei LED herausgesucht werden, bei denen sich der Leuchtpunkt "oben" befindet.



Abbildung 5: PCB des DIP-Schalters und der RGB-LED in Vergrößerung

Die beiden rechten LED sind ebenfalls 4-polig, jedoch wird "Grün" nicht verwendet und der Pin abgeschnitten. Daher sind nur 3 Pads vorhanden.

Gemeinsame Anode (+)  
Common Anode

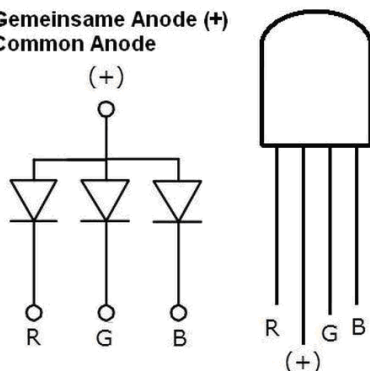


Abbildung 6: Pin-Zuordnung der RGB-LED des M041

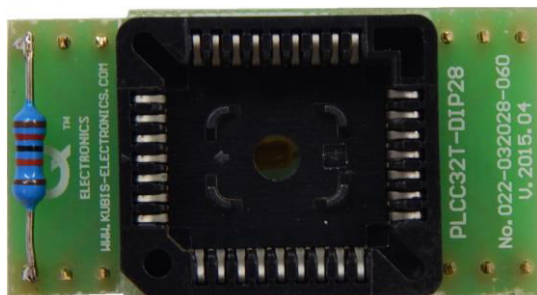
Die Ausrichtung der LED ergibt sich aus dem Bestückungsaufdruck (flache Seite). Bei den 4-poligen LED empfiehlt es sich, diese vor dem Anlöten abzuwinkeln (vorher die notwendige Länge zum Abwinkeln ausmessen).

Die LED haben folgende Werte in der Einheit mcd: Rot 800-100, Grün 2000-4000, Blau 1500-2000. Der Vorwärtsstrom beträgt 20 mA maximal und 10  $\mu$ A minimal. Nur für diese LED sind die Widerstandswerte gültig.



Mehr muß nicht beachtet werden. Das Modul sollte fertig aufgebaut sofort betriebsbereit sein.

Da EEPROM-Schaltkreise in der DIP-Ausführung, welche das /BSY-Signal am Pin 1 zur Verfügung stellen, nicht mehr so einfach erhältlich sind, bietet sich als Alternative ein EEPROM-IC in PLCC-Ausführung an. Dazu bedarf es eines Adapters, der in der folgenden Abbildung dargestellt ist.



**Abbildung 7: Adapterplatine DIP-PLCC zur Verwendung im M041**

Die M041-Platine ist nicht dafür ausgelegt, eine PLCC-DIP-Adapterplatine aufzunehmen. Daher muß die kleine Adapterplatine so schmal wie möglich sein, damit die umliegenden Bauteile problemlos eingesetzt werden können. Hier ist die Fertigkeit des Anwenders gefragt (Verwendung einer Feile).

Zu beachten ist, daß die beiden Bauteile (Kondensator, Widerstand), welche im Normalfall unterhalb der IC-Fassung angebracht werden müssen, hier auf die Adapterplatine gelötet werden.

Der Abblockkondensator wird dabei auf der Lötseite angebracht (hier nicht im Bild).

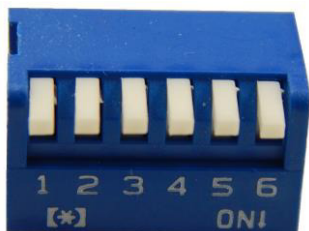
Des weiteren ist es wichtig, daß die Bauhöhe der Adapterplatine 12 mm nicht übersteigt, weil dies die maximale lichte Höhe im Inneren eines Modulgehäuses ist. Das bedeutet, daß keine normalen IC-Adapterstiftleisten verwendet werden dürfen. Der Durchmesser eines Stiftes der verwendeten Stiftleisten darf maximal 0,65 mm betragen.

### 5.3 Inbetriebnahme und Benutzung des M041

Durch die Neu- und Weiterentwicklung des Programms EEPSPD<sup>1</sup> gestaltet sich die Inbetriebnahme des M041 sehr einfach.

Zweckmäßigerweise wird das Modul in Schacht 8 gesteckt, denn nur dieser ist von vornherein ausgelegt, Module mit Kennbyte 01 automatisch beim Systemstart zu aktivieren und ein darauf befindliches Programm zu starten. Zum Programmieren, d.h. Beschreiben der EEPROM sollte das Kennbyte F8 eingestellt sein.

Nachfolgend eine Abbildung des DIP-Schalters zum Einstellen des Kennbytes mit anschließender Erläuterung der Schalterstellungen.



**Abbildung 8: DIP-Schalter zum Einstellen des Kennbytes**

Die Schalter 1 bis 4 dienen dabei zur Einstellung des Kennbytes.

Die Schalter 5 und 6 des DIP-Schalters werden für die Aktivierung des Hardware-Schreibschutzes der beiden Submodule verwendet.

DIP-SW	1	2	3	4	mögliche Kennbytes
01	on	on	-	on	Start-ROM (M033)
F1	-	on	-	on	16k CMOS-RAM/EEPROM (M122)
F8	-	-	on	on	16k EPROM (M028)
FC	-	-	on	-	16k ROM (M006)

**Tabelle 3: Schalterstellungen des DIP-Schalters zum Einstellen des Kennbytes**

Natürlich kann das M041 auch in jedem anderen Schacht unter Beachtung der Modulprioritätskette betrieben werden.

<sup>1</sup> Das Programm EEPSPD wurde 1997 von Mario Leubner für EEPROM-IC entwickelt und im Jahr 2017 von ihm grundlegend erweitert, so daß eine komfortable "in system"-Programmierung aller im KC-System vorhandenen EEPROM-IC ermöglicht wird.



Das Programm EEPSPD stellt nach dem Laden 4 Kommandos im CAOS-Menü zur Verfügung.

```
* KC-CAOS 4.7 USB *
/BASIC
/REBASIC
/SWITCH
/JUMP
/MENU
/SAVE
/VERIFY
/LOAD
/COLOR
/DISPLAY
/MODIFY
/WINDOW
/KEY
/LSIDEV
/V24DUP
/DEVICE
/PAGESIZE
/EEPCLR
/EEPSPD
/SDP
/
/PAGESIZE
Pagesize = 20
/EEPCLR AADR EADR+1 (Steckplatz)
/EEPSPD Quelle Ziel Länge (Steckplatz)
        ohne Steckplatz kein SDP!
/SDP Steckplatz 0/1
/
/EEPCLR 4000 8000
LASTADR: 7FFF
/
```

Abbildung 9: EEPSPD-Kommandos / Hilfe und Aufruf

Jeder EEPROM hat eine Maximalkapazität von 8 KB. Beide EEPROM eines Submoduls zusammen 16 KB.

## PAGESIZE

Die meisten EEPROM-Typen, so auch die zum gelieferten Bausatz gehörenden, ermöglichen die Programmierung im Blockmodus. Dabei werden Daten in sogenannten "pages" zu je 16, 32 oder 64 Byte in den EEPROM geschrieben. Es können auf dem M041 auch EEPROM eingesetzt werden, welche nur den Einzelbyte-Modus beim Programmieren beherrschen.

Das Kommando PAGESIZE dient dazu, die korrekte Größe eines Blockes einzustellen. Dabei sind die Werte hexadezimal anzugeben. Ein Wert von 1 bedeutet dabei Einzelbyte-Modus.

### SDP

Es gibt EEPROM, welche mit einer speziellen Programmiersequenz in einen schreibgeschützten Zustand versetzt werden können. Dies sind zum Beispiel der CAT28C65B von Catalyst und der KM28C65A von Samsung. Mit dem Kommando SDP kann der Schreibschutz gesetzt bzw. wieder aufgehoben werden. Dazu ist neben dem gewünschten Zustand auch die Modulsteckplatzadresse anzugeben. Zu beachten ist hierbei, daß sich die Zustandsänderung auf beide EEPROM eines Submoduls auswirkt. Die Anwendung von SDP ist nur sinnvoll bei EEPROM, welche dieses Merkmal auch aufweisen. Bei anderen EEPROM kann es unter Umständen zu einer Korruption des Speicherinhaltes kommen.

### EEPCLR

Hierbei handelt es sich um das explizite Löschen des EEPROM bzw. eines Teilbereiches, in dem für jedes Byte der Wert FF eingetragen wird. Da moderne Programmiergeräte für EEPROM eine separate Löschfunktion zur Verfügung stellen, wurde diese auch in EEPSPD integriert.

Das Kommando erwartet zwei hexadezimale Wertangaben: Den Beginn und das Ende+1 des zu löschenden Bereiches.

### EEPSPD

Dieses Kommando ermöglicht die Programmierung des EEPROM bzw. eines Teilbereiches. Es werden drei Parameter erwartet und eine Option kann gesetzt werden. Die drei Parameter sind hexadezimale Wertangaben und bestimmen die Startadresse des zu programmierenden Inhaltes, die Zieladresse auf dem EEPROM und die Anzahl der zu programmierenden Bytes. Die Option ist ebenfalls ein Hexadezimalwert, die Steckplatzadresse des Moduls. Wird diese angegeben, dann wird bei SDP-fähigen EEPROM der Schreibschutz vor dem Beschreiben aufgehoben und anschließend wieder aktiviert.

```

* KC-CAOS 4.7 USB *
%BASIC
%REBASIC
%SWITCH
%JUMP
%MENU
%SAVE
%VERIFY
%LOAD
%COLOR
%DISPLAY
%MODIFY
%WINDOW
%KEY
%LSTDEV
%V24DUP
%DEVICE
%PAGESIZE
%EEPCLR
%EEPSDP
%SDP
%SW 3 0
03 FF 00
%SW 10 83
10 F4 83
%EEPSDP 4000 8000 4000
%LASTADR: 7FFF BFFF
%
```

Abbildung 10: EEPSDP / Programmierung eines EEPROM ohne SDP

Sowohl EEPCLR als auch EEPSDP führen eine Überprüfung durch, ob die Programmierung korrekt erfolgte. Die Überprüfung richtet sich nach der eingestellten PAGESIZE, d.h. genau so viele Bytes werden während eines Prüfzyklusses getestet. Schlägt der Test fehl, wird der Vorgang des Löschens bzw. Programmierens abgebrochen und die letzte korrekte Anfangsadresse einer geprüften "page" ausgegeben.

Bis auf das Kommando PAGESIZE erfordern alle Kommandos, daß sich der EEPROM programmieren, d.h. beschreiben läßt. Daher muß sichergestellt sein, daß zum einen der Hardwareschreibschutz aufgehoben ist und zum anderen das Modul mit gesetztem Schreibschutzbit im Steuerbyte aktiviert wurde.

Bis auf das Kommando SDP - es existiert keine Möglichkeit zu überprüfen, ob der Schreibschutz korrekt aktiviert bzw. deaktiviert wurde - geben alle Kommandos im Fehlerfall aussagekräftige Meldungen aus.

Das folgende Beispiel zeigt, wie die gesamten 16 KB eines Submoduls in einem Zug programmiert werden können.

- LOAD -> EEPSPD
- RAM 4 aktivieren: SWITCH 4 3
- Daten nach 4000H bis 7FFFH laden: LOAD 8000 (falls die Ladeadresse der Datei C000H ist) -> DATEN.KCC. Diese Datei benötigt den KCC-Vorspann, d.h. die Angabe der Ladeadresse.
- Alternative nur unter CAOS 4.7: LOAD-Funktion des M030 verwenden. Damit können einfache BIN- bzw. ROM-Dateien geladen werden. Das LOAD-Kommando wird dabei auf das eingestellte DEVICE (DISK, USB) umgeleitet.
- IRM abschalten: SWITCH 1 0
- RAM 8 abschalten: SWITCH 3 0
- M041 einschalten: SWITCH STECKPLATZ 83 für Submodul 0 oder SWITCH STECKPLATZ+1 83 für Submodul 1. Dabei darauf achten, daß der Hardware-Schreibschutz nicht aktiviert ist.
- EEPROM programmieren: EEPSPD 4000 8000 4000 [STECKPLATZ]
  1. Parameter: Quellstartadresse. 2. Parameter: Zielstartadresse. 3. Parameter: Länge in Bytes.
  4. Parameter: SDP aufheben und nach dem Programmieren wieder aktivieren. Wichtig dabei: Nur bei EEPROM angeben, die auch SDP unterstützen, sonst schlägt die Programmierung fehl.

Wurde ein Autostartprogramm programmiert und befindet sich das Modul im Steckplatz 8, dann kann das M041 anschließend auf das Kennbyte "01" gesetzt werden. Zuvor ist das Modul mit "SW STECKPLATZ 0" auszuschalten. Nach einem Reset des KC sollte das Programm starten.

## 6 Anwendungsfall / Programmerstellung

Wie schon weiter oben beschrieben, war die Motivation für den Nachbau des M040 ein kleines Programm, welches vor einiger Zeit entwickelt wurde und welches als Autostartprogramm genutzt werden sollte. Dieses Programm wird im Folgenden vorgestellt. Dabei sind vielleicht auch einige nützliche Hinweise dabei.

Die Ur-Fassung des Programms ist lauffähig unter CAOS 4.4 und höher. Benötigt wird ein USB-Modul. Das Programm ist eine kleine Diashow verschiedener Bilder, die auf Ausflügen und in Urlauben in ganz Deutschland entstanden. Die Bilder liegen natürlich im JPEG-Format vor und müssen zunächst KC85-tauglich gemacht werden. Dazu wurde in PERL ein eigener Algorithmus implementiert, welcher das Bild in der Größe und Farbtiefe (s/w) reduziert. Ursprünglich wurden diese Bilder über das USB-Speichermedium eingelesen und dann Pixel für Pixel am KC dargestellt.

Da sich mit der Veröffentlichung von CAOS 4.7 ganz andere Möglichkeiten ergeben, wurde das Programm grundlegend geändert.

Es ist nun ausschließlich unter CAOS 4.7 lauffähig und setzt ein USB-Speichermedium oder eine CAOS-formatierte Diskette voraus. Der EEPROM-Inhalt des USB-Moduls muß aktualisiert werden, um die neuen CAOS-Funktionen nutzen zu können.

Diese neuen und vom Programm verwendeten Funktionen sind das Laden von Daten per LOAD-Kommando direkt vom USB-Speichermedium oder der Diskette und das Auflisten des Inhaltes eines Verzeichnisses des aktiven Speichermediums per DIR-Kommando. Des weiteren wird eine Funktion benutzt, die erst ab CAOS 4.6 zur Verfügung steht: Das Setzen des aktiven DEVICE per SETDEV.

Die Urfassung des Programms war in purem JKCEMU<sup>2</sup>-Basic programmiert. Der Emulator stellt eine komplette und leistungsstarke Entwicklungsumgebung zur Verfügung, die den Vergleich mit einem kommerziellen Produkt nicht scheuen muß.

Auch die Neufassung ist in diesem Basic-Dialekt geschrieben, jedoch ist es eine Symbiose zwischen Basic und Assembler (das Basic erlaubt die Verwendung von Assembler-Funktionen bzw. -Anweisungen). Assembler deshalb, weil viele Eigenschaften des neuen Programms nicht anders umgesetzt werden können.

---

<sup>2</sup> Der Emulator JKCEMU ist eine Entwicklung von Jens Müller.

Was zeichnet das Programm aus?

Zunächst einmal ist es autostartfähig<sup>3</sup>. Damit ein Programm autostartfähig ist, müssen folgende Voraussetzungen erfüllt sein.

- Das Programm muß in einem ROM-Speicher abgelegt sein.
- Dieser ROM-Speicher muß sich auf einem Modul befinden, welches sich im Steckplatz 8 befindet und das Kennbyte "01" aufweist (zum Beispiel das M041).
- CAOS blendet beim Start des Systems und dem Vorhandensein eines Autostart-Moduls den darauf befindlichen Speicher ab der Adresse 4000H ein. Der RAM 4 wird dabei zuvor automatisch abgeschaltet. Dann wird auf die Adresse 4000H gesprungen und dort muß lauffähiger Code stehen.
- Das heißt, das Programm muß auf die Adresse 4000H gelinkt sein.

Da der ROM-Speicher des Moduls natürlich nur lesbar ist, muß an der Adresse 4000H eine Umladeroutine stehen. Diese transportiert das komplette Programm, exklusive der Umladeroutine, an die Adresse 0300H und springt anschließend auf diese Adresse. Die ersten 32 Byte werden dabei nur beim Autostart ausgeführt und schalten das Modul im Schacht 8 wieder ab und aktivieren den RAM 4 wieder (schreibfähig). Anschließend wird auf die Startadresse des eigentlichen Programms gesprungen.

Das bedeutet, daß das eigentliche Programm auf die Adresse 0320H gelinkt sein muß. Damit steht das Programm dann auch nach Beendigung als Menü-Kommando zur Verfügung.

Das Programm selbst zeigt zunächst einen Startbildschirm an, da es initialisiert werden muß. Wird es per Autostart gestartet, wird eine kleine Melodie abgespielt, welche thematisch zu den angezeigten Bildern paßt. Beim Aufruf aus dem Menü heraus kann per Parameter mit dem Wert "1" das Abspielen der Melodie unterbunden werden.

Standardmäßig wird ein USB-Speichermedium vorausgesetzt und verwendet. Sollte dieses nicht vorhanden sein, wird ein einmaliger Versuch unternommen, eine CAOS-formatierte Diskette als Speichermedium zu verwenden. Schlägt auch das fehl, beendet sich das Programm.

Ist ein USB-Speichermedium vorhanden, werden darauf zum einen die Bilddaten erwartet und zum anderen eine Initialisierungsdatei. Ist eine INI-Datei vorhanden, werden damit die fest-codierten Parameter zur Anzahl der Bilder, zur Anzeigedauer der Bilder, zum Bestimmen, ob eine Melodie abgespielt werden soll und zur Lautstärke der Melodie überschrieben. Beim Betrieb von Diskette werden die fest-codierten Parameter verwendet.

---

<sup>3</sup> Rolf Weidlich und Mario Leubner gaben dazu wertvolle Tips.

Das Vorhandensein der Bilddaten wird nicht geprüft, bis auf die Ausnahme des Startbildschirmes. Ist ein erwartetes Bild nicht vorhanden, wird das Programm demzufolge "hart" abstürzen.

Die kleine Diashow läuft endlos und zeigt die Bilder in einer zufallsgesteuerten Reihenfolge an. Rund 3 Sekunden vor dem Anzeigen des nächsten Bildes blinkt im unteren, rechten Bildschirmbereich ein weißer Kreis. Während dieser Zeit kann durch das Betätigen einer beliebigen Taste das Programm beendet werden.

Das Programm nutzt zum Laden der Bilddaten den RAM 4 und lädt dessen Inhalt dann zum Anzeigen in den RAM 8; ab Adresse 8000H mit einer Länge von 2600 Bytes. Dies entspricht genau der Größe eines CAOS-Bildschirmes (320x256 Bildpunkte, 81920 Pixel, pro Byte 8 Pixel, 2600 Bytes). Da die Bilder schwarz-weiß sind, genügt ein einfaches Laden der Daten in den RAM 8.

Wird das Programm beendet, werden der RAM 4 geleert, die Bildschirme aufgeräumt und zum CAOS-Menü zurückgesprungen.



Abbildung 11: Startbildschirm und eines der Bilder des Programms SRNSHOW

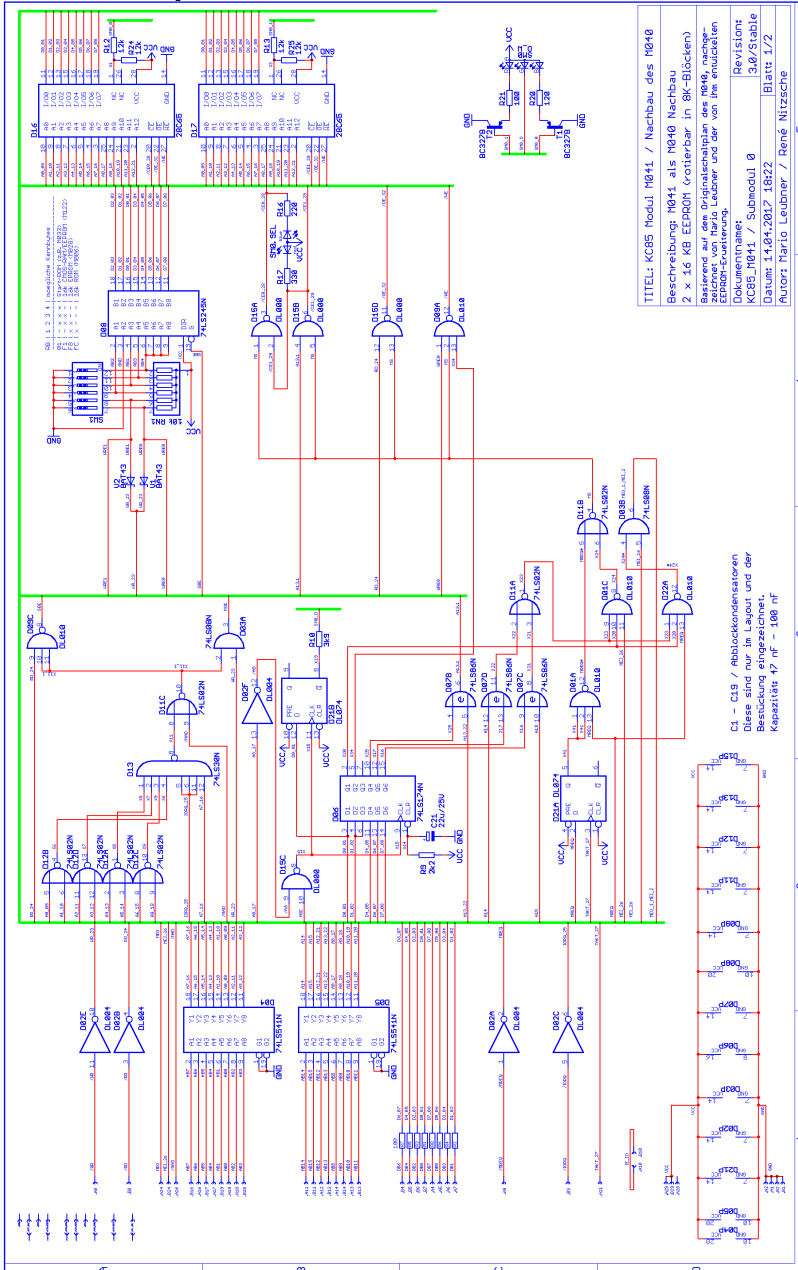
Damit das Laden der Bilddaten korrekt abläuft ist es unbedingt notwendig, daß ein im System vorhandenes M030-Modul (EPROMMER) hinter dem USB-Modul (M052) steckt.

Warum das so ist, kann nicht vollständig erklärt werden. Da aber beide Module mit ihrem Festwertspeicher auf die Basisadresse C000H eingelenket werden, kann durch das Beschreiben des RAM 4 während des Programmablaufs der EPROM-Inhalt des M030 aktiviert werden, wodurch das M052 ausgeblendet wird.

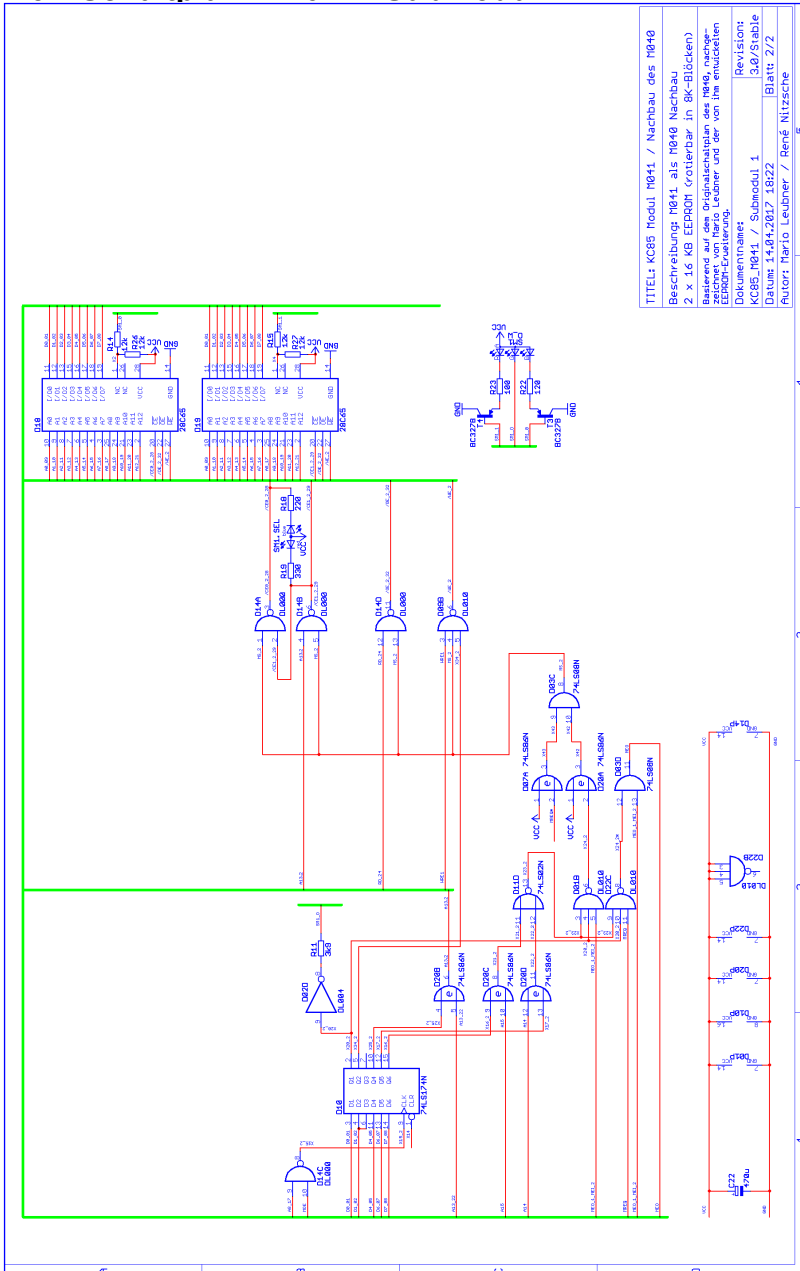




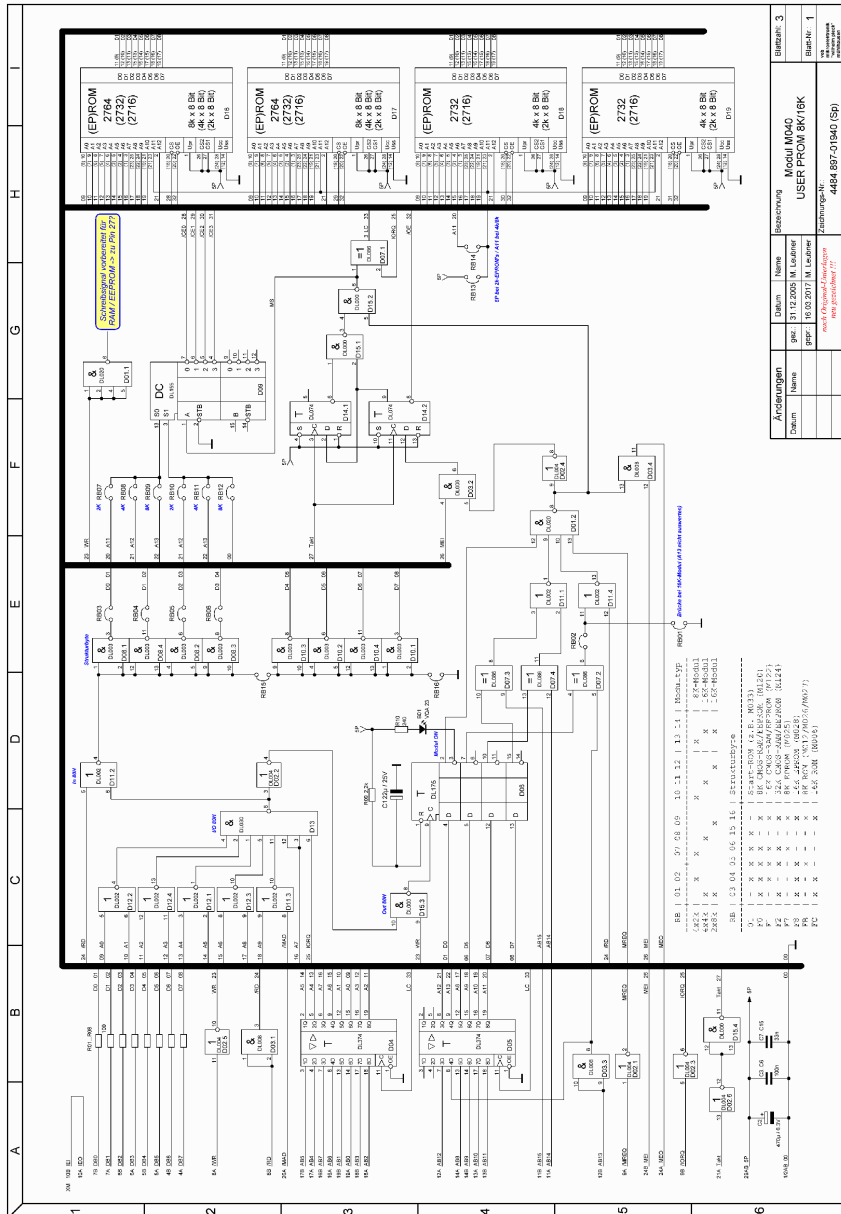
## 7.2 Schaltplan – M041 / Submodul 0



## 7.3 Schaltplan – M041 / Submodul 1



## 7.4 Schaltplan – M040



## 8 Anlage B – Bestückungsliste

Bauteilnummer	Bezeichnung	Wert
C1-C9, C11-C20, C22 <sup>4</sup>	Kondensator	100 nF
C21	Kondensator	22 µF / 25 V
C22	Kondensator	470 µF / 10 V
R1 - R8	Widerstand	100
R9	Widerstand	2k2
R10	Widerstand	3k9
R11	Widerstand	3k9
R12 - R15	Widerstand	12k
R16	Widerstand	220
R17	Widerstand	330
R18	Widerstand	220
R19	Widerstand	330
R20	Widerstand	120
R21	Widerstand	100
R22	Widerstand	120
R23	Widerstand	100
R24 - R27	Widerstand	12k
RN1	Widerstands-Netzwerk	10k
SW1	DIP-Schalter	DP-Bauform
T1 - T4	PNP-Transistor	BC327B
V1, V2	Schottky-Diode	BAT43
SM0	RGB-LED	common anode
SM0.	RGB-LED	common anode
SM1	RGB-LED	common anode
SM1.	RGB-LED	common anode

<sup>4</sup> Aufgrund eines Fehlers ist der C22 zweimal in der Stückliste enthalten

Bauteilnummer	Bezeichnung	Wert
D01	Schaltkreis	DL010D / 74LS10N
D02	Schaltkreis	DL004D
D03	Schaltkreis	74LS08N / DL008D
D04, D05	Schaltkreis	74LS541N
D06	Schaltkreis	74LS174N
D07	Schaltkreis	74LS86N
D08	Schaltkreis	74LS245N
D09	Schaltkreis	DL010D / 74LS10N
D10	Schaltkreis	74LS174N
D11, D12	Schaltkreis	74LS02N / DL002D
D13	Schaltkreis	74LS30N
D14, D15	Schaltkreis	DL000D
D16 - D19	Schaltkreis	28C65
D20	Schaltkreis	74LS86N
D21	Schaltkreis	DL074D
D22	Schaltkreis	74LS10N
IC-Fassung	DIL 14	13 Stück
IC-Fassung	DIL 16	2 Stück
IC-Fassung	DIL 20	3 Stück
IC-Fassung	DIL 28-6	4 Stück
Platine	Platine	Platine

## 9 Anlage C – Quellcode SRNSHOW

```

REM -----
REM Deklarationen

DECLARE SUB ZEICHNE_SANDUHR
DECLARE SUB LADE_BILD(BNR)
DECLARE SUB LADE_RAM8
DECLARE SUB SPIELE_MUSIK
DECLARE SUB SETZE_V1_V2_LS(V1_V2, LS)
DECLARE SUB SETZE_Z1_Z2(Z1_Z2)
DECLARE SUB SETZE_TD(TD)
DECLARE SUB GIB_TON_AUS
DECLARE FUNCTION KEINE_MUSIK_SPIELEN
DECLARE FUNCTION AKTIVIERE_DEV(DEVICE)
DECLARE SUB CLEANUP
DECLARE SUB CLEAN_RAM4

REM -----
REM Globale Variablen.
NO_FIRST=0
MAX_BILDNR=25
REM Damit ergeben sich insgesamt 10 Sekunden. 5 Sekunden Ladezeit
REM plus 5 Sekunden Wartezeit. Jeder Einerschritt == Eine Sekunde.
ANZEIGEDAUER=5
SPIELEMUSIK=0
KEINEMUSIKSPIELEN=KEINE_MUSIK_SPIELEN
TONDAUER1=20
TONDAUER2=40
LAUTSTAERKE=26
DUMMYRET=0
LESEINI=0
INITFAIL=1
INITUSB=1
INITDEV=0
CANCEL$=""
REM -----

REM -----
REM Start des Programms
REM -----
REM Hauptprogramm
  REM Hintergrundinitialisierung fuer beide Bildschirme (0,1)
  PAPER BLACK : INK BLACK : CLS
  PRINT CHR$(27);"3"
  PAPER BLACK : INK BLACK : CLS
  PRINT CHR$(27);"2"

  MOVE 88,220 : COLOR WHITE,BLACK : LABEL "SRN PROUDLY PRESENTS!"
  MOVE 88,220 : LABEL "SRN PROUDLY PRESENTS!"
  MOVE 10,190 : LABEL "'Kein scho" : PSET 64,197 : PSET 66,197
  LABEL "ner Land - Eine Reise durch Deutschland'"

```

```
REM Sanduhr anzeigen lassen
ZEICHNE_SANDUHR

MOVE 25,60
LABEL "Programm wird initialisiert ... Bitte warten Sie."

PRINT CHR$(27);"4"

REM USB-Aktivierung und Ueberpruefung, ob Wartebild vorhanden.
INITDEV=AKTIVIERE_DEV(0)

REM Wenn USB-Aktivierung fehlgeschlagen ist, wird versucht auf
REM Disk-Betrieb umzustellen.
IF INITDEV<>1 THEN
    INITUSB=0
    INITFAIL=AKTIVIERE_DEV(1)
    REM Disk-Betrieb nicht moeglich. Versuch DISK zu starten.
    IF INITFAIL<>1 THEN
        IF INITFAIL=2 THEN
            PRINT CHR$(27);"2"
            MOVE 15,40
            LABEL "USB-Initialisierung fehlgeschlagen ... Starte DISK"
            PRINT CHR$(27);"4"
            PAUSE 20
            REM Einmaliger Versuch des Starts des CAOS-DISK-Betriebs.
            CLS
            ASM CODE " LD A, 0FCH"
            ASM CODE " CALL 0F003H"
            ASM CODE " DEFB 27H"
        ENDIF
        PRINT CHR$(27);"2"
        MOVE 15,25
        LABEL "Initialisierung fehlgeschlagen ... Ende mit <ENTER>"
        PRINT CHR$(27);"4"
        INPUT CANCEL$
        CLS
        CLEANUP
    ENDIF
ENDIF

CLS

REM Quit-Taste zuruecksetzen.
QUIT=1
DO
    POKE 509, 0
    QUIT=PEEK(509)
LOOP UNTIL QUIT=0

REM INI-Datei lesen. Jedoch nur, wenn USB aktiv ist.
IF INITUSB THEN
    OPEN "V:SRNSHOW.INI" FOR INPUT AS #1
    IF ERR=0 THEN
```

```

    LESEINI=1
ENDIF

IF LESEINI=1 THEN
    INPUT #1, MAX_BILDNR
    INPUT #1, ANZEIGEDAUER
    INPUT #1, SPIELEMUSIK
    INPUT #1, LAUTSTAERKE
    CLOSE #1
ENDIF
ENDIF

REM Wartemeldung zeichnen lassen

LADE_BILD(999)
ZEICHNE_SANDUHR

PRINT CHR$(27);"3" : CLS

IF SPIELEMUSIK=1 THEN
    REM Wenn das Programm per Menuebefehl aufgerufen wurde,
    REM dann kann das Abspielen der Musik per Paramter (1)
    REM deaktiviert werden.
    IF KEINEMUSIKSPIELEN<>1 THEN
        SPIELE_MUSIK
    ENDIF
ENDIF
ENDIF

REM Aktueller Zeichenbildschirm
ZBS=1
REM Diashow
QUIT=0
DO
    IF ZBS=1 THEN
        PRINT CHR$(27);"3"
        ZBS=0
    ELSE
        PRINT CHR$(27);"4"
        ZBS=1
    ENDIF
    PAPER BLACK : INK BLACK : CLS

    INK WHITE

    LADE_BILD(RND(MAX_BILDNR))
    IF NO_FIRST=1 THEN
        FOR C=1 TO ANZEIGEDAUER
            PAUSE 10
        NEXT C
    ENDIF

    IF ZBS=0 THEN
        PRINT CHR$(27);"1"
    
```



```

ELSE
  PRINT CHR$(27);"2"
ENDIF
INK WHITE+BLINKING
FOR C=1 TO 3
  CIRCLE 314,5,C
NEXT C

IF NO_FIRST=1 THEN
  FOR C=1 TO 15000
    QUIT=PEEK(509)
    IF QUIT<>0 THEN
      PAUSE 1
      EXIT FOR
    ENDIF
  NEXT C
ELSE
  NO_FIRST=1
ENDIF
LOOP UNTIL QUIT<>0

REM Aufraeumen.
CLEANUP

END
REM -----

REM -----
REM Sanduhr zeichnen.
SUB ZEICHNE_SANDUHR

  INK WHITE
  LINE 150,125,170,125 : LINE 149,126,171,126 :LINE 150,125,150,117
  LINE 149,126,149,117 : LINE 150,117,157,110 :LINE 149,117,156,110
  LINE 157,110,150,103 : LINE 156,110,149,103 :LINE 150,103,150,95
  LINE 149,103,149,94  : LINE 150,95,170,95  :LINE 149,94,171,94
  LINE 170,95,170,103  : LINE 171,94,171,103  :LINE 170,103,163,110
  LINE 171,103,164,110 : LINE 163,110,170,117 :LINE 164,110,171,117
  LINE 170,117,170,125 : LINE 171,117,171,126 :LINE 153,120,167,120
  LINE 153,119,167,119 : LINE 153,118,167,118 :LINE 154,117,166,117
  LINE 155,116,165,116 : LINE 156,115,164,115 :LINE 157,114,163,114
  LINE 158,113,162,113 : LINE 159,112,161,112 :PSET 160,111
  LINE 155,96,165,96   : LINE 156,97,164,97   :LINE 157,98,163,98
  LINE 158,99,162,99   : LINE 159,100,161,100 :PSET 160,101
  PSET 161,109 : PSET 159,107 : PSET 161,106 : PSET 158,106
  INK WHITE+BLINKING
  PSET 160,104 : PSET 156,103 : PSET 166,102
  INK WHITE
  PSET 157,101 : PSET 153,97 : PSET 167,98 : PSET 163,105

END SUB
REM -----

```

```
REM -----
REM Bild laden.
SUB LADE_BILD(BNR)

    DEF USR0 = ASM(" LD HL,MYLOAD")

    DUMMYRET=USR0 (VAL (STR$ (BNR) ,16))
    GOTO ENDE_LADE_BILD

    ASM CODE "MYLOAD:"

    ASM CODE " PUSH DE"

    REM Einer-Stelle.
    ASM CODE " LD HL,FILENAME+7"
    ASM CODE " LD A, E"
    ASM CODE " AND 0FH"
    ASM CODE " LD E, A"
    ASM CODE " LD A, 30H"
    ASM CODE " ADD E"
    ASM CODE " LD (HL), A"
    REM Zehner-Stelle.
    ASM CODE " POP DE"
    ASM CODE " PUSH DE"
    ASM CODE " LD HL,FILENAME+6"
    ASM CODE " LD A, E"
    ASM CODE " RRA"
    ASM CODE " RRA"
    ASM CODE " RRA"
    ASM CODE " RRA"
    ASM CODE " AND 0FH"
    ASM CODE " LD E, A"
    ASM CODE " LD A, 30H"
    ASM CODE " ADD E"
    ASM CODE " LD (HL), A"
    REM Hunderter-Stelle.
    ASM CODE " POP DE"
    ASM CODE " PUSH DE"
    ASM CODE " LD HL,FILENAME+5"
    ASM CODE " LD A, D"
    ASM CODE " AND 0FH"
    ASM CODE " LD E, A"
    ASM CODE " LD A, 30H"
    ASM CODE " ADD E"
    ASM CODE " LD (HL), A"

    ASM CODE " POP DE"

    ASM CODE " LD HL,FILENAME"
    ASM DATA "FILENAME: DB 'SRNB_XXX.KCC',0"

    ASM CODE " LD BC, 0H"
    ASM CODE " LD (0B781H), BC"
```

```
ASM CODE " CALL 0F003H"
ASM CODE " DEFB 10H"

ASM CODE " RET"

ENDE_LADE_BILD:

REM Bild anzeigen.
LADE_RAM8

END SUB
REM -----

REM -----
REM RAM8 laden.
SUB LADE_RAM8

CLS

ASM CODE " LD HL, 4000H"
ASM CODE " LD DE, 8000H"
ASM CODE " LD BC, 2800H"
ASM CODE " LDIR"

END SUB
REM -----

REM -----
REM Musik abspielen.
SUB SPIELE_MUSIK

LOCAL I

FOR I=1 TO 2
  SETZE_V1_V2_LS(0, LAUTSTAERKE)
  SETZE_Z1_Z2(171)
  SETZE_TD(TONDAUER1)
  GIB_TON_AUS
  GIB_TON_AUS
  GIB_TON_AUS
  SETZE_V1_V2_LS(0, LAUTSTAERKE+2)
  SETZE_Z1_Z2(128)
  SETZE_TD(TONDAUER2)
  GIB_TON_AUS
  SETZE_V1_V2_LS(0, LAUTSTAERKE)
  SETZE_Z1_Z2(102)
  GIB_TON_AUS
  SETZE_Z1_Z2(114)
  SETZE_TD(TONDAUER1)
  GIB_TON_AUS
  SETZE_Z1_Z2(128)
  GIB_TON_AUS
  SETZE_V1_V2_LS(0, LAUTSTAERKE+2)
```

```

    SETZE_Z1_Z2(114)
    SETZE_TD(TONDAUER2)
    GIB_TON_AUS
    PAUSE 10
NEXT I
FOR I=1 TO 2
    SETZE_V1_V2_LS(0, LAUTSTAERKE)
    SETZE_Z1_Z2(102)
    SETZE_TD(TONDAUER1)
    GIB_TON_AUS
    SETZE_Z1_Z2(128)
    GIB_TON_AUS
    SETZE_Z1_Z2(114)
    GIB_TON_AUS
    SETZE_V1_V2_LS(0, LAUTSTAERKE+2)
    SETZE_Z1_Z2(102)
    GIB_TON_AUS
    SETZE_V1_V2_LS(0, LAUTSTAERKE)
    SETZE_Z1_Z2(86)
    GIB_TON_AUS
    SETZE_Z1_Z2(96)
    GIB_TON_AUS
    SETZE_Z1_Z2(102)
    GIB_TON_AUS
    SETZE_Z1_Z2(114)
    GIB_TON_AUS
    SETZE_Z1_Z2(128)
    GIB_TON_AUS
    SETZE_V1_V2_LS(0, LAUTSTAERKE+2)
    SETZE_Z1_Z2(114)
    GIB_TON_AUS
    SETZE_V1_V2_LS(0, LAUTSTAERKE)
    SETZE_Z1_Z2(96)
    GIB_TON_AUS
    SETZE_Z1_Z2(102)
    GIB_TON_AUS
    SETZE_Z1_Z2(114)
    GIB_TON_AUS
    SETZE_V1_V2_LS(0, LAUTSTAERKE+2)
    SETZE_Z1_Z2(128)
    GIB_TON_AUS
    IF I=2 THEN
        SETZE_V1_V2_LS(0, LAUTSTAERKE)
        SETZE_Z1_Z2(136)
        GIB_TON_AUS
        SETZE_TD(TONDAUER2)
        SETZE_Z1_Z2(128)
        GIB_TON_AUS
    ELSE
        SETZE_V1_V2_LS(0, LAUTSTAERKE)
        SETZE_Z1_Z2(114)
        GIB_TON_AUS
        SETZE_TD(TONDAUER2)

```

```

        SETZE_Z1_Z2(102)
        GIB_TON_AUS
    ENDIF
    PAUSE 10
    NEXT I

END SUB
REM -----

REM -----
REM Vorbereitung Tonausgabe: ARG1+1, ARG2+1, ARG3 setzen.
SUB SETZE_V1_V2_LS(V1_V2, LS)

    DEF USR3 = ASM(" LD HL,SET_V1_V2")
    DUMMYRET=USR3(V1_V2)
    DEF USR6 = ASM(" LD HL,SET_LS")
    DUMMYRET=USR6(LS)

    GOTO ENDE_SETZE_V1_V2_LS

    ASM CODE "SET_V1_V2:"
    ASM CODE " LD A, E"
    ASM CODE " LD (0B783H), A"
    ASM CODE " LD (0B785H), A"
    ASM CODE " RET"

    ASM CODE "SET_LS:"
    ASM CODE " LD A, E"
    ASM CODE " LD (0B786H), A"
    ASM CODE " RET"

    ENDE_SETZE_V1_V2_LS:

END SUB
REM -----

REM -----
REM Vorbereitung Tonausgabe: ARG1 und ARG2 setzen.
SUB SETZE_Z1_Z2(Z1_Z2)

    DEF USR2 = ASM(" LD HL,SET_Z1_Z2")
    DUMMYRET=USR2(Z1_Z2)

    GOTO ENDE_SETZE_Z1_Z2

    ASM CODE "SET_Z1_Z2:"
    ASM CODE " LD A, E"
    ASM CODE " LD (0B782H), A"
    ASM CODE " LD (0B784H), A"
    ASM CODE " RET"

    ENDE_SETZE_Z1_Z2:

```

```

END SUB
REM -----

REM -----
REM Vorbereitung Tonausgabe: ARG3+1 setzen.
SUB SETZE_TD(TD)

    DEF USR7 = ASM(" LD HL,SET_TD")
    DUMMYRET=USR7(TD)

    GOTO ENDE_SETZE_TD

    ASM CODE "SET_TD:"
    ASM CODE " LD A, E"
    ASM CODE " LD (0B787H), A"
    ASM CODE " RET"

    ENDE_SETZE_TD:

END SUB
REM -----

REM -----
REM Ausgabe des Tones.
SUB GIB_TON_AUS

    ASM CODE " CALL 0F003H"
    ASM CODE " DEFB 35H"

END SUB
REM -----

REM -----
REM Bei Aufruf per Menuebefehl kann das Abspielen der Musik per
REM Parameter (1) unterbunden werden.
FUNCTION KEINE_MUSIK_SPIELEN

    DEF USR9 = ASM(" LD HL,CHECKARG")
    KEINE_MUSIK_SPIELEN=USR9(0)

    GOTO ENDE_KEINE_MUSIK_SPIELEN

    ASM CODE "CHECKARG:"
    ASM CODE " LD DE, (0B781H)"
    ASM CODE " LD A, E"
    ASM CODE " LD HL, (0B782H)"
    ASM CODE " AND A"
    ASM CODE " LD BC, 0H"
    ASM CODE " LD (0B781H), BC"

    ASM CODE " RET"

    ENDE_KEINE_MUSIK_SPIELEN:

```

```
END FUNCTION
REM -----

REM -----
REM Prueft auf das Vorhandensein des USB-Sticks durch DIR-Aufruf
REM und Suche der Datei fuer das Wartebild.
FUNCTION AKTIVIERE_DEV(DEVICE)

    LOCAL I
    LOCAL DEV
    LOCAL DEVFOUND
    DEV=1
    DEVFOUND=0

    IF DEVICE=1 THEN
        GOTO AKTIVIERE_DISK
    ENDIF

    DEF USR5 = ASM(" LD HL,SETUSB")

    REM A900H + dezimal 36 -> Erster Buchstabe des 1. DEVICE.
    FOR I=36 TO 192 STEP 32
        IF USR5(I)=1 THEN
            DEVFOUND=1
            EXIT FOR
        ENDIF
        DEV=DEV+1
    NEXT I

    IF DEVFOUND=0 THEN
        AKTIVIERE_DEV=2
        GOTO ENDE_AKTIVIERE_DEV
    ENDIF

    GOTO AKTIVIERE_DEVICE

    AKTIVIERE_DISK:

    DEF USR1 = ASM(" LD HL,SETDISK")

    REM A900H + dezimal 36 -> Erster Buchstabe des 1. DEVICE.
    FOR I=36 TO 192 STEP 32
        IF USR1(I)=1 THEN
            DEVFOUND=1
            EXIT FOR
        ENDIF
        DEV=DEV+1
    NEXT I

    IF DEVFOUND=0 THEN
        AKTIVIERE_DEV=2
        GOTO ENDE_AKTIVIERE_DEV
    ENDIF
```

```
AKTIVIERE_DEVICE:

REM DEVICE auf USB oder DISK setzen.
DEF USR4 = ASM(" LD HL,SETDEV")
DUMMYRET=USR4 (DEV)

GOTO CHECK_DEV

ASM CODE "SETDEV:"
ASM CODE " LD A, E"
ASM CODE " CALL 0F003H"
ASM CODE " DEFB 49H"

ASM CODE " RET"

ASM CODE "SETUSB:"

ASM DATA "COMPAREUSB: DB 'USB'"
ASM CODE " LD HL, COMPAREUSB"
ASM CODE " LD D, 0A9H"
ASM CODE " LD B, 3"
ASM CODE "V1_USB: LD A, (DE)"
ASM CODE " CP (HL)"
ASM CODE " JR NZ, NOTSETUSB"
ASM CODE " INC HL"
ASM CODE " INC DE"
ASM CODE " DJNZ V1_USB"

ASM CODE " LD HL, 0001H"
ASM CODE " RET"

ASM CODE "NOTSETUSB: LD HL, 0000H"
ASM CODE " RET"

ASM CODE "SETDISK:"

ASM DATA "COMPAREDISK: DB 'DISK'"
ASM CODE " LD HL, COMPAREDISK"
ASM CODE " LD D, 0A9H"
ASM CODE " LD B, 4"
ASM CODE "V1_DISK: LD A, (DE)"
ASM CODE " CP (HL)"
ASM CODE " JR NZ, NOTSETDISK"
ASM CODE " INC HL"
ASM CODE " INC DE"
ASM CODE " DJNZ V1_DISK"

ASM CODE " LD HL, 0001H"
ASM CODE " RET"

ASM CODE "NOTSETDISK: LD HL, 0000H"
ASM CODE " RET"
```



```

CHECK_DEV:

DEF USR8 = ASM(" LD HL,CHECKDEV")
AKTIVIERE_DEV=USR8 (DEVICE)

GOTO ENDE_AKTIVIERE_DEV

ASM CODE "CHECKDEV:"

ASM CODE " LD A, 0H"
ASM CODE " CP E"
ASM CODE " JR NZ, DISK"
ASM CODE " LD DE,USBACTIVATE"
ASM DATA "USBACTIVATE: DB 'SRNB_999.KCC',0"
ASM CODE " JR CHECKFILE"
ASM CODE "DISK: LD DE,DISKACTIVATE"
ASM DATA "DISKACTIVATE: DB 'SRNB_999KCC',0"
ASM CODE "CHECKFILE: LD HL, 0H"
ASM CODE " CALL 0F021H"
ASM CODE " DEFB 8H"

ASM CODE " RET"

ENDE_AKTIVIERE_DEV:

END FUNCTION
REM -----
REM -----
REM Speicher aufräumen, Bildschirme leeren, Ruecksprung zu CAOS.
SUB CLEANUP

REM Aufräumen
IF ZBS=1 THEN
    PRINT CHR$(27);"4"
ELSE
    PRINT CHR$(27);"3"
ENDIF

REM RAM4 wieder leer hinterlassen.
CLEAN_RAM4

PAPER BLUE : INK WHITE : CLS
PRINT CHR$(27);"1"
PAPER BLUE : INK WHITE : CLS

REM Ruecksprung zu CAOS mit MENU-Anzeige
ASM CODE " LD A, 12"
ASM CODE " CALL 0F003H"
ASM CODE " DEFB 0"
ASM CODE " CALL 0F003H"
ASM CODE " DEFB 46H"

```

```
END SUB
REM -----

REM -----
REM RAM4 leeren.
SUB CLEAN_RAM4

    ASM CODE " LD HL, 4000H"
    ASM CODE " LD DE, 4001H"
    ASM CODE " LD BC, 27FFH"
    ASM CODE " LD (HL), 0"
    ASM CODE " LDIR"

END SUB
REM -----
```



# mikroelektronik



**RFT**



Alte Computer by SRN – <http://www.kc-und-atari.de>