

# AVR-ChipBasic2: Interna

## V1.49 (c) 2006-2014 Jörg Wolfram

### 1 Organisation des Videospeichers

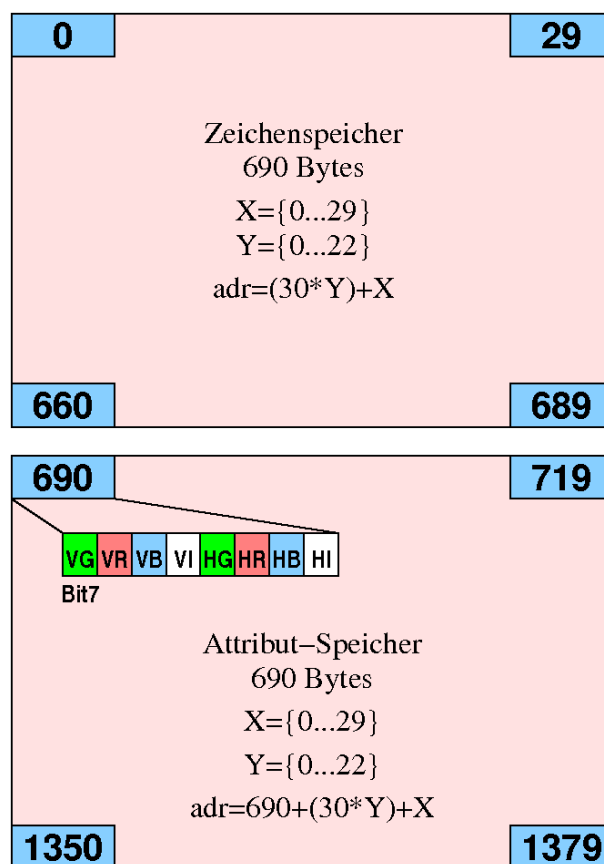
#### 1.1 Unterschiede bei 8 und 16 Farben

Die I (Intensitäts-) Bits sind nur von Relevanz, wenn die 16-Farben-Erweiterung angeschlossen ist. Die etwas ungewöhnliche Verteilung der Bits (Intensität als LSB) ist durch die Kompatibilität zum 8-Farben Modus der "normalen" Hardware bedingt.

#### 1.2 Videomode 0 (Standard-Mode)

Dieser Mode ist ein Textmodus mit festem Zeichensatz von 256 Zeichen. Jedem Zeichen kann getrennt Vorder- und Hintergrundfarbe zugewiesen werden. Es gibt 23 Zeilen a 30 Zeichen, dies resultiert einfach daraus daß das Projekt zum Teil mit einem 37 cm TV-Gerät mit recht großem Rand entwickelt wurde. Im Bildspeicher finden sich nacheinander:

Anfangsadresse	Endadresse	Bytes	Funktion
0	689	690	Zeichen
690	1379	690	Attribute
1380	2759	1380	Backup für Monitor

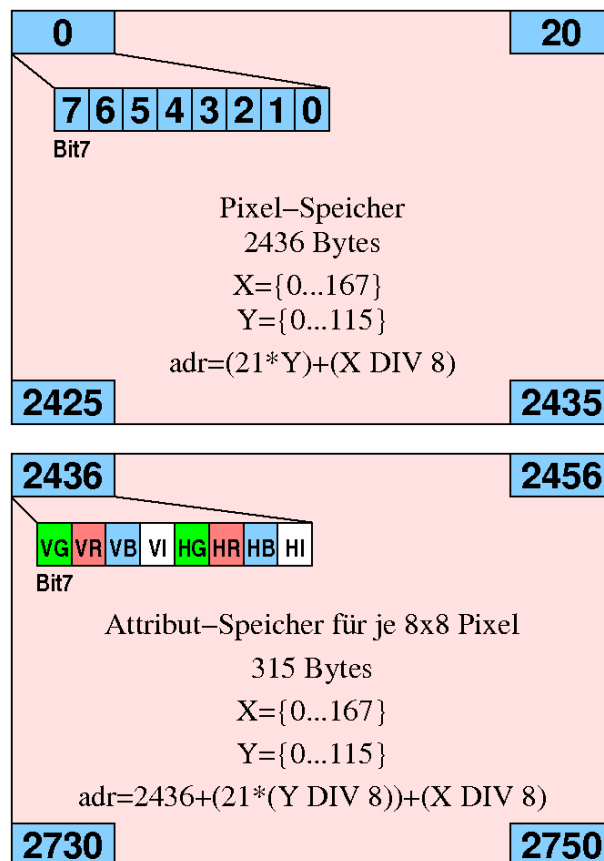


Für die Pseudografik sind die Zeichen 0...15 in 4 „Pixel“ aufgeteilt. Bei 23 Zeilen a 30 Zeichen ergibt sich so eine Arbeitsfläche von 60x46 Punkten.

### 1.3 Videomode 1

Die Pixelauflösung beträgt 168x116 Pixel, wobei für jeweils 8x8 (am unteren Rand nur 4x8) Pixel Vorder- und Hintergrundfarbe eingestellt werden können.

Anfangsadresse	Endadresse	Bytes	Funktion
0	2435	2436	Pixelinformation
2436	2750	315	Attribute
2751	2759	9	ungenutzt

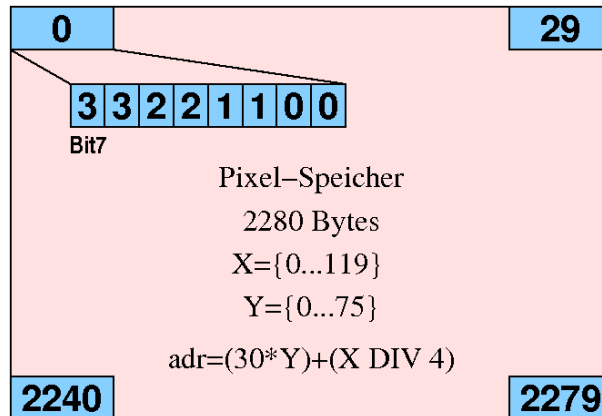


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entspricht Bit 0 dem Pixel ganz links.

## 1.4 Videomode 2

Die Pixelauflösung beträgt 120x76 Pixel, jedes Pixel kann eine aus 4 über die Palette einstellbaren Farben annehmen.

Anfangsadresse	Endadresse	Bytes	Funktion
0	2279	2280	Pixelinformation
2280	2759	480	ungenutzt

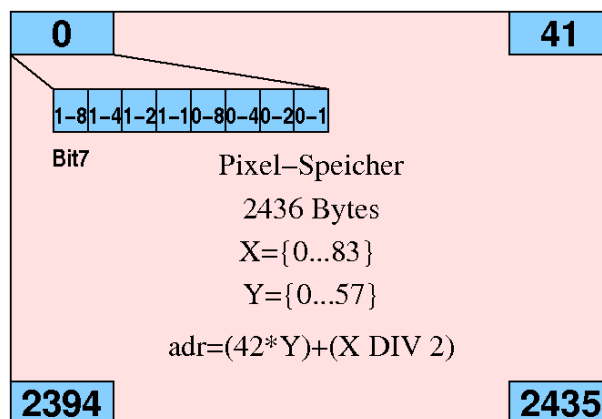


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entsprechen Bit 0 und 1 dem Pixel ganz links. Jedes Pixel wird durch zwei Bits repräsentiert, die auf einen der ersten 4 Paletteneinträge zeigen. Der Paletteneintrag bestimmt dann die Farbe.

## 1.5 Videomode 3

Die Pixelauflösung beträgt 84x58 Pixel, jedes Pixel kann eine aus 16 über die Palette einstellbaren Farben annehmen.

Anfangsadresse	Endadresse	Bytes	Funktion
0	2435	2436	Pixelinformation
2436	2759	324	ungenutzt

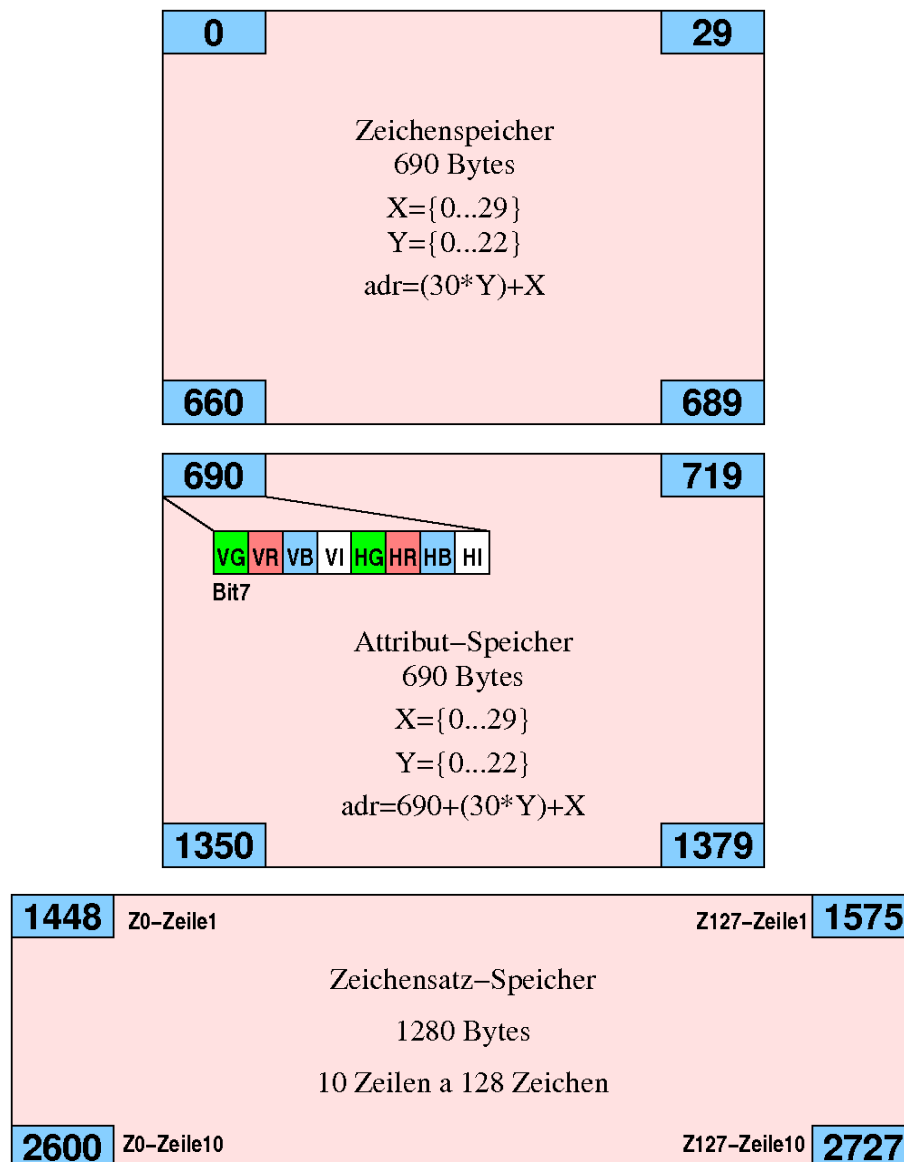


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Jedes Byte ist in 2 Nibbles zu 4 Bit unterteilt, die auf einen der 16 Paletteneinträge zeigen. Der Paletteneintrag bestimmt dann die Farbe.

## 1.6 Videomode 4

Dieser Mode ist ein Textmodus mit variablen Zeichensatz von 128 Zeichen. Jedem Zeichen kann getrennt Vorder- und Hintergrundfarbe zugewiesen werden. Beim Wechsel in den Videomodus werden die unteren 128 Zeichen aus dem festen Zeichensatz in den variablen Zeichensatz kopiert. Im Bildspeicher finden sich nacheinander:

Anfangsadresse	Endadresse	Bytes	Funktion
0	689	690	Zeichen
690	1379	690	Attribute
1448	2727	1280	Zeichensatz
2728	2759	32	ungenutzt



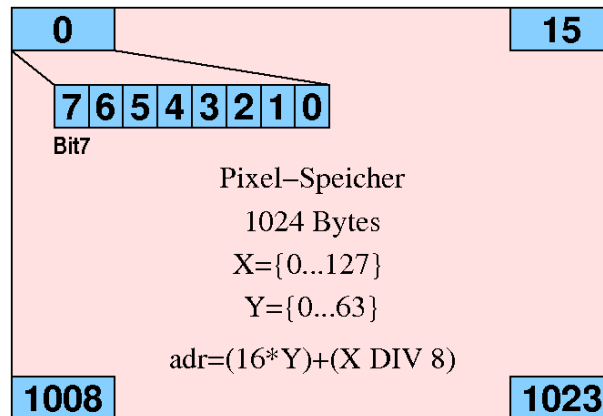
Im Zeichensatz sind die Bytes nach Zeichenzeilen geordnet, also 128 Bytes für die erste Zeichenzeile, dann 128 Bytes für die zweite Zeichenzeile...

Dabei sind die Pixel innerhalb einer Zeichenzeile noch differentiell gespeichert, Bit 7 ist das Pixel ganz links, ist Bit 6 "0", hat das nächste Pixel die gleiche Farbe, bei einer "1" ändert sie sich. Ein Byte ergibt eine Zeichenzeile von 6 Pixeln wobei Bit 0 und 1 nicht berücksichtigt werden, ein Zeichen besteht aus 10 dieser Zeichenzeilen.

## 1.7 Videomode 5

Die Pixelauflösung beträgt 128x64 Pixel, jedes Pixel kann eine aus 2 über die Palette einstellbaren Farben annehmen. Dieser Modus dient hauptsächlich zur Emulation von Grafik-LCD mit der entsprechenden Auflösung.

Anfangsadresse	Endadresse	Bytes	Funktion
0	1023	1024	Pixelinformation
1024	2759	1736	Backup für Monitor

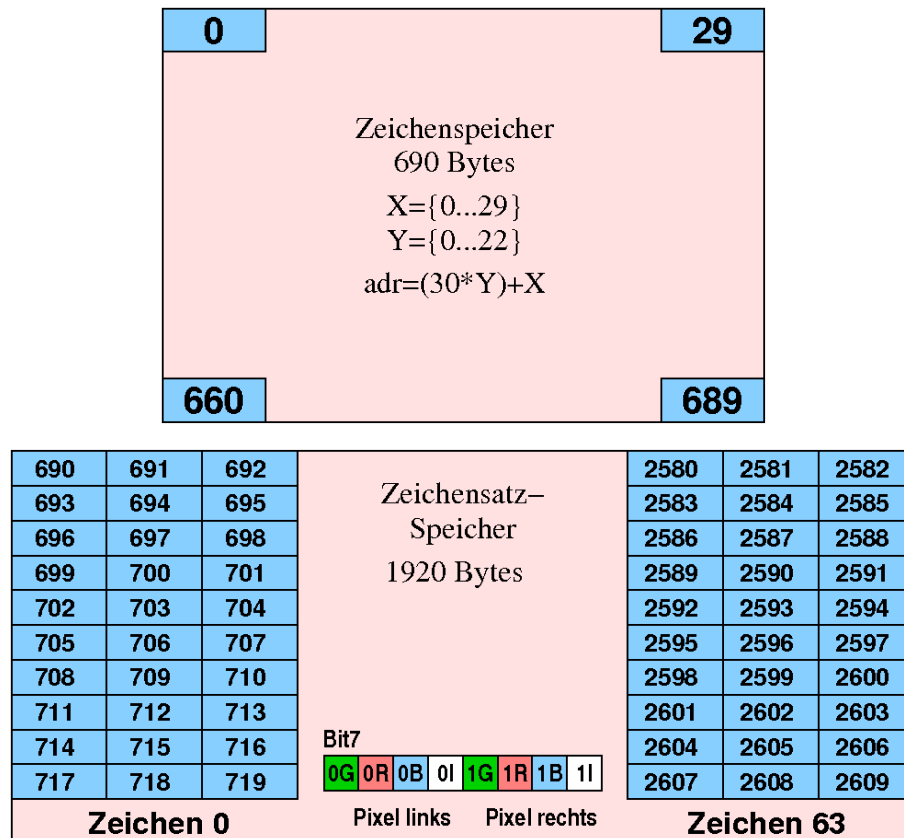


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entspricht Bit 0 dem Pixel ganz links. Für jedes Pixel gibt es genau ein Bit, welches auf einen der ersten beiden Paletteneinträge zeigt. Der Paletteneintrag bestimmt dann die Farbe.

## 1.8 Videomode 6

Dieser Modus ist ein Textmodus mit variablen Zeichensatz von 64 Zeichen. Jedem Pixel in jedem Zeichen kann eine der 8 Farben zugewiesen werden. Beim Wechsel in den Videomodus werden die unteren Zeichen aus dem festen Zeichensatz in den variablen Zeichensatz kopiert. Im Bildspeicher finden sich nacheinander:

Anfangsadresse	Endadresse	Bytes	Funktion
0	689	690	Zeichen
690	2609	1920	Zeichensatz
2610	2759	150	ungenutzt



Im Zeichensatz liegen die 30 Bytes jedes Zeichens direkt hintereinander. Jedes Pixel wird durch drei Bits repräsentiert, innerhalb des Bytes entsprechen Bit 1...3 dem rechten und Bit 5...7 dem linken Pixel. Drei Bytes ergeben eine Zeichenzeile von 6 Pixeln, ein Zeichen besteht aus 10 dieser Zeichenzeilen.

## 1.9 Videomode 7

Der Videomodus 7 wurde in der aktuellen Version entfernt. Stattdessen kann ein eigener Treiber auf Programmplatz 8 installiert werden. Für Video Treiber ist der ID-Bereich 0xc0 bis 0xdf im HIGH-Byte reserviert. Ist keine derartige ID vorhanden, bewirkt ein **VMODE 7** nur einen Constant Error. Bei Binärprogrammen muss allerdings selbst darauf geachtet werden, ob ein passender Treiber installiert ist.

## 2 Das API für Binärprogramme

### 2.1 Allgemeines

Hmmm... , API - was ist das? API ist die Abkürzung für "Application Programming Interface" und dient in diesem fall dazu, auch native (z.B. mit einem Assembler erzeugte) Binärprogramme auf dem ChipBasic2 Computer laufen zu lassen.

### 2.2 Möglichkeiten und Einschränkungen

Da wäre zuerst die Programmgröße. Diese ist auf 3072 Bytes begrenzt, wobei noch am Anfang des Speicherbereiches 12 Bytes für den Programmnamen, 1 Byte für die Unterscheidung zu BASIC Programmen (ist bei Binärprogrammen immer "N") und 19 weitere Bytes für z.B. das Programm-Icon reserviert sind.

Damit die Programme auch nach einem Update des Systems noch funktionieren, sollten direkte Unterprogrammaufrufe (ohne Umweg über die API Funktionen) vermieden werden. Ebenso sollten die Auskunfts-Funktionen genutzt werden, um die Adressen von bestimmten Speicherbereichen abzufragen, anstelle mit den gerade aktuellen Adressen zu arbeiten. Außerdem ist es nicht sinnvoll, innerhalb des Programmes mit **JMP** oder **CALL** zu arbeiten, da die Programm-Adresse je nach Programmplatz variiert. Damit ist leider auch nicht so einfach, auf z.B. im Quelltext eingebunde Tabellen zuzugreifen. Allerdings ist das über das API möglich:

```
.include      "M644Pdef.inc"
#include      "api.inc"

.org          0x4000

start:        .db "TestprogrammN",0xec,0xff,0xff
              .db "+--+"
              .db "ABCD"
              .db "+--+"

.org          0x4010

lese:         call  api_getvram                ;setzt Y auf den Anfang des Bildspeichers
              ldi   XL,LOW(daten-start)        ;Bestimmung des Offsets
              ldi   XH,HIGH(daten-start)
              call  api_dataptr                ;setzt Z auf die Adresse von "daten:"
              call  api_orom                   ;ROM-Text ab (Z) ausgeben
              ret

daten:        .db "Mein Text",0
```

Am Anfang müssen 12 bytes Programmname und ein "N" (0x4E) stehen, danach folgen 3 Bytes für verschiedene Flags und Bibliotheks/Treibernummer. Im Anschluss stehen 12 Bytes für das Programm-Icon zur Verfügung. Das eigentliche Programm beginnt mit einem Offset von 32 Bytes (16 Words).

Anstelle die API-Funktionen über **CALL** oder **JMP** aufzurufen ist es praktischer, die Makros zu verwenden. Allerdings funktionieren die nicht mit allen Assemblern richtig (nur mit AVRA getestet):

```
.include      "M644Pdef.inc"
.include      "api_macros.inc"

.org          0x4000

start:        .db "TestprogrammN",0xec,0xff,0xff
               .db "+--+"
               .db "ABCD"
               .db "+--+"

.org          0x4010

lese:         api_getvram                ;setzt Y auf den Anfang des Bildspeichers
               ldi XL,LOW(daten-start)   ;Bestimmung des Offsets
               ldi XH,HIGH(daten-start)
               api_dataptr                ;setzt Z auf die Adresse von "daten:"
               api_orem                   ;ROM-Text ab (Z) ausgeben
               ret

daten:        .db "Mein Text",0
```

Der mit dem Assembler erzeugte Code ist aber letztendlich derselbe.

## 2.3 Die Flags im Header

In Byte 13 des Headers befinden sich verschiedene Flags und die Vordergrundfarbe des Icons.

Bit	Name	Funktion
7	—	z.Zt. nicht benutzt
6	UPAR	0 = Treiber benutzt Parallelport
5	ISLIB	0 = Programm ist eine Bibliothek
4	EXEC	0 = Programm ist aus dem Hauptemü heraus startbar
3...0	ICOL	ICON-Vordergrundfarbe

Bei der ICON-Vordergrundfarbe kann theoretisch jeder der 16 Werte benutzt werden, für Übersichtlichkeit und Konsistenz innerhalb des Systems sollte bei der Entwicklung neuer Programme die folgende Zuordnung beibehalten werden:

Farbcode	Farbe	Dateityp
0000	schwarz	nicht benutzt (unsichtbar)
x001	blau	nicht benutzt (mangelnder Kontrast)
x010	rot	Reserviert für Loader
x011	magenta	Vorgabe für <b>Treiber</b>
x100	grün	Vorgabe für <b>Bibliotheken</b>
x101	cyan	Vorgabe für <b>Binärprogramme</b>
x110	gelb	Reserviert für nicht startbare BASIC-Programme (Text)
x111	weiss	Reserviert für BASIC-Programme



Nachfolgend einige Beispiele für sinnvolle Byte-Kombinationen:

Typ	Flagbyte	Programmtyp
"B"	0xf7	Daten
"N"	0xe5	Binärprogramm
"N"	0xd4	Binär-Bibliothek
"N"	0xf3	Treiber
"N"	0xb3	Treiber, benutzt Parallelport
"L"	0xf2	Loader

In Byte 14 befinden sich weitere 8 Flags. Diese stehen für verschiedene Erweiterungen. Bietet ein Programm bzw. eine Bibliothek eine oder mehrere Erweiterungen, so ist das entsprechende Bit auf 0 gesetzt. Beim Systemstart und wenn ein Programm geladen oder gelöscht wird, werden die Flagbytes aller Programme eingelesen und das Programm mit dem letzten Auftreten in einer Tabelle gespeichert. Diese Tabelle kann dann benutzt werden, um die Funktionen aufzurufen. Wenn es keine entsprechende Erweiterung im Speicher gibt, ist das entsprechende Byte = 0xff, ansonsten entspricht es der High-Adresse in Words des letzten Programmes mit aktivem Flag. Befinden sich zum Beispiel auf den Programmplätzen 4 und 6 Video-Treiber, so steht im Tabelleneintrag 0 der Wert 0x66 für die Anfangsadresse von Programmplatz 6. Mit der API Funktion **api\_getprg** wird das Y Register auf den Anfang der Tabelle gesetzt. Wird aus einem Binärprogramm heraus der Inhalt des Programmspeichers (Programm 1-8) modifiziert, sollte vorher die Funktion **api\_extdisable** aufgerufen werden. Diese sperrt die externen Funktionen, die zyklisch aufgerufen werden (Videoausgabe, Sound, Frame) um zu verhindern dass Programmaufrufe zu ungültigen Routinen ausgeführt werden. Anschließend muß mit einem Aufruf von **api\_extsearch** die Tabelle wieder aktualisiert werden.

Die Belegung der Bits geht aus folgender Tabelle hervor, die Bitnummer entspricht dabei gleichzeitig dem Offset gegenüber dem Tabellenanfang:

Bit	Funktion
7	Nicht belegt
6	Programm enthält einen Font
5	Programm enthält einen Dateisystemtreiber
4	Programm enthält Treiber für externen/internen Speicher
3	Programm enthält BASIC-Erweiterung
2	Programm enthält Sound-Routinen
1	Programm enthält zyklische Routine, die in jedem Frame aufgerufen wird
0	Programm enthält Video-Treiber

In Byte 15 steht die Bibliotheksnummer. Bei Programmen, die keine Bibliotheksfunktion bereitstellen sollte diese mit 0xff belegt sein. Mit dem BASIC Befehl **LFIND** kann dann festgestellt werden, ob sich die gesuchte Bibliothek im Speicher befindet.

## 2.4 Beispiele für Programmköpfe

Die nachfolgenden Beispiele enthalten Programmköpfe für verschiedene Programmtypen. Wenn Programme, Bibliotheken oder Treiber zusätzliche Funktionen bereitstellen, müssen die zugehörigen Bits im Flagbyte 2 gelöscht sein.

### 2.4.1 Programmkopf von einfachen Binärprogrammen

Binärprogramme werden normalerweise vom Hauptmenü aus gestartet, alternativ ist auch ein Start über XCALL P,n möglich, wobei P hier für den Programmplatz 1...8 steht und n nicht ausgewertet wird.

WORD Offset	BYTE Offset	Länge (Bytes)	Funktion
0x00	0x00	12	Programmname
—	0x0c	1	”N” als Kennzeichnung für Binärprogramme
—	0x0d	1	Flagbyte 1, normalerweise 0xe5
—	0x0e	1	Flagbyte 2, normalerweise 0xff
—	0x0f	1	Bibliotheksnnummer, sollte 0xff sein
0x08	0x10	12	Icon-Daten
0x0e	0x1c	2	kann beliebig verwendet werden
0x0f	0x1e	2	kann beliebig verwendet werden
0x10	0x20	2	Start des Programms

#### 2.4.2 Programmkopf von Binärbibliotheken

Die einzelnen Funktionen von Binärbibliotheken werden vom BASIC aus gestartet, es können entweder numerische Parameter oder auch ein String übergeben werden. Wie die Parameterübergabe erfolgt, wird durch das T-Flag festgelegt. Ist es gelöscht, sind numerische Parameter übergeben worden. Bei gesetztem T-Flag befindet sich in Y die Adresse des Strings im RAM. Das Ende des Strings ist entweder durch ”Gänsefüßchen” (0x22) oder durch 0xff markiert.

WORD Offset	BYTE Offset	Länge (Bytes)	Funktion
0x00	0x00	12	Programmname
—	0x0c	1	”N” als Kennzeichnung für Binärprogramme
—	0x0d	1	Flagbyte 1 = 0xd4
—	0x0e	1	Flagbyte 2 = 0xff oder Zusatzfunktionen
—	0x0f	1	Bibliotheksnnummer
0x08	0x10	12	Icon-Daten
0x0e	0x1c	2	kann beliebig verwendet werden
0x0f	0x1e	2	kann beliebig verwendet werden
0x10	0x20	2	Sprung zur Routine Nr.0
0x11	0x22	2	Sprung zur Routine Nr.1
0x12	0x24	2	Sprung zur Routine Nr.2, weitere Routinen bis 119 möglich

### 2.4.3 Programmkopf von BASIC-Erweiterungen

Das BASIC kann mit BASIC-Erweiterungen um eigene Befehle ergänzt werden. Wird dann ein externer Befehl (beginnt mit Unterstrich) erkannt, so wird die Parser-Routine aufgerufen.

WORD Offset	BYTE Offset	Länge (Bytes)	Funktion
0x00	0x00	12	Programmname
—	0x0c	1	”N” als Kennzeichnung für Binärprogramme
—	0x0d	1	Flagbyte 1 = 0xf4
—	0x0e	1	Flagbyte 2 = 0xf7
—	0x0f	1	Bibliotheksnummer
0x08	0x10	12	Icon-Daten
0x0e	0x1c	2	sollte 0xffff sein
0x0f	0x1e	2	sollte 0xffff sein
0x10	0x20	2	(Sprung zur) Parser-Routine
0x11	0x22	2	Sprung zur Routine Nr.1
0x12	0x24	2	Sprung zur Routine Nr.2, weitere Routinen bis 119 möglich

#### 2.4.4 Programmkopf von Video-Treibern

Nicht benutzte Routinen sollten mit einem RET beendet werden.

WORD Offset	BYTE Offset	Länge (Bytes)	Funktion
0x00	0x00	12	Programmname
—	0x0c	1	”N” als Kennzeichnung für Binärprogramme
—	0x0d	1	Flagbyte 1 = 0xd3
—	0x0e	1	Flagbyte 2 = 0xfe
—	0x0f	1	Bibliotheksnummer
0x08	0x10	12	Icon-Daten
0x0e	0x1c	2	Sprung zur IN-expansion oder R24 (ereg) auf 40 setzen
0x0f	0x1e	2	Sprung zur OUT-expansion oder R24 (ereg) auf 40 setzen
0x10	0x20	2	Sprung zur (Demo) Anwendung falls Programm startbar
0x11	0x22	2	Sprung zur INIT Videomode (T-Flag gesetzt) oder EXIT Videomode (T-Flag gelöscht)
0x12	0x24	2	Sprung zur CLS-Routine
0x13	0x26	2	Sprung zur Zeichenausgabe, Zeichen in R20
0x14	0x28	2	Sprung zur Routine, um Cursorposition auf XH:XL setzen
0x15	0x2a	2	Sprung zur Plot XH:XL
0x16	0x2c	2	Sprung zur Newline-Routine, Cursor zum Anfang der nächsten Zeile
0x17	0x2e	2	Sprung zur Videoausgabe-Routine
0x18	0x30	2	Sprung zur Init-Routine bei Systemstart
0x19	0x32	2	Sprung zur Routine Nr.9, kann bis 119 gehen

#### 2.4.5 XMEM Erweiterungen

Mittels Speichererweiterungen kann der nutzbare Speicher vergrößert werden. Maximal sind 64 Kilobytes zusätzlicher Speicher möglich, welcher Seitenweise in die Arrayzellen 768...1023 eingeblendet wird. Im Flagbyte muss dazu Bit 4 gelöscht sein.

WORD Offset	BYTE Offset	Länge (Bytes)	Funktion
0x19	0x32	2	Sprung zur Array-READ Routine (Page/R20=Pointer, XL=Daten)
0x1a	0x34	2	Sprung zur Array-WRITE Routine (Page/R20=Pointer, XL=Daten)
0x1b	0x36	2	Sprung zur Byte-READ Routine (X+) -> r18
0x1c	0x38	2	Sprung zur Byte-WRITE Routine r18 -> (X+)
0x1d	0x3a	2	Sprung zur Word-READ Routine (X+) -> r19/r18
0x1e	0x3c	2	Sprung zur Word-WRITE Routine r19/r18 -> (X+)
0x1f	0x3e	2	Sprung zur XMEM-CHEXK Routine (X=Adresse der letzten Speicherzelle oder 0, falls kein XMEM existiert)
0x20	0x40	2	Sprung zur XMEM-CLEAR Routine (beschreibt komplettes XMEM mit 0x00)
0x21	0x42	2	Sprung zur Routine Nr.17, kann bis 119 gehen

#### 2.4.6 Frame Interrupts

Mittels der Frame-Interrupt Erweiterung kann eine eigene Routine am unteren Ende des farbigen Randes (Border) aufgerufen werden. Anwendungsbeispiel wäre z.B. die Abfrage einer Echtzeituhr oder die Abtastung von DCF77 Signalen. Im Flagbyte 2 muss dazu Bit 1 gelöscht sein.

WORD Offset	BYTE Offset	Länge (Bytes)	Funktion
0x21	0x42	2	Sprung zur Frame-INT Routine

### 2.4.7 SOUND Erweiterungen

Mittels Sound-Erweiterungen kann die Tonausgabe in gewissen Grenzen an eigene Bedürfnisse angepasst werden. So können eigene Tabellen für die Tonerzeugung verwendet und die Lautstärkeberechnung modifiziert werden. Im Flagbyte 2 muss dazu Bit 2 gelöscht sein.

WORD Offset	BYTE Offset	Länge (Bytes)	Funktion
0x22	0x44	2	Sprung zur Envelope Routine (am Ende des Frames)
0x23	0x46	2	Sprung zur Noten Routine

## 2.5 Übergabe von Parametern und Resultaten

Um Erweiterungen für das BASIC zu schreiben, kann auf die Übergabeparameter und den Resultatwert zugegriffen werden. Dies geschieht mit der Funktion **api\_getvalues**, die einen Zeiger auf die folgende Struktur im Y-Register zurückliefert:

Offset	Format	Inhalt
0	INTEGER	Rückgabewert der Funktion
2	INTEGER	Aufruf-Paramter 1
4	INTEGER	Aufruf-Paramter 2
6	INTEGER	Aufruf-Paramter 3
8	INTEGER	Aufruf-Paramter 4
10	INTEGER	Aufruf-Paramter 5
12	BYTE	Anzahl der übergebenen Paramter

Die Parameter-Werte haben nur Gültigkeit wenn das T-Flag gelöscht ist (numerische Parameter), der Rückgabewert ist aber in beiden Fällen nutzbar.

## 2.6 Adressbereich für I/O-Erweiterungen

Damit I/O Erweiterungen möglichst universell genutzt werden können, ist eine Standardisierung zumindest für die wichtigsten Adressbereiche notwendig. Die vorliegende Tabelle stellt nur eine Minimalbasis dar und wird wahrscheinlich im Laufe der Zeit noch ergänzt werden müssen. Alle Treiber müssen für Adressbereiche die sie nicht bedienen den Wert 40 (0x28) im Register R24 (ereg) zurückliefern.

Startadresse	Endadresse	Bedeutung
0x800	0x809	Pegel der nutzbaren Ausgabe-Pins, wenn I/O deaktiviert ist (0/1)
0x810	0x819	Datenrichtung der nutzbaren Ausgabe-Pins, wenn I/O deaktiviert ist (0/1)
0x900	0xbff	Treiberspezifische I/O
0x0c00	0xffff	zur Zeit nicht belegt

## 2.7 Build-Prozess zum Erstellen einer eigenen Applikation

Das Erstellen eines Binärprogrammes unterscheidet sich etwas von der "normalen" Vorgehensweise. Die nachfolgende Aufzählung beschreibt das Vorgehen unter Linux und wird so unter anderen Betriebssystemen nur eingeschränkt oder nicht funktionieren. Die notwendigen Include-Dateien für die von mir erstellten Programme und Bibliotheken werden im Verzeichnis **/usr/local/include/chipbasic2** erwartet, ggf. muss der Pfad angepasst werden.

Folgende Schritte sind zum Erstellen einer Binärdatei notwendig:

1. Erstellung des Programmes
2. Assemblierung mit **AVRA**
3. Umwandlung in das Binärformat mit **hex2bin**
4. Kontrolle, dass die Binärdatei maximal 3072 Bytes groß ist
5. Erstellung einer beliebigen Datei, welche zusammen mit der Binärdatei aus dem vorigen Schritt genau 3072 Bytes lang ist
6. Die erstellte Datei an die Binärdatei anhängen
7. Die im Schritt 5 erzeugte Datei kann nun wieder gelöscht werden

Um das nicht jedesmal von Hand machen zu müssen gibt es ein kleines Build-Script, welches einen Großteil der Arbeit abnimmt. Aufgerufen wird es mit **build-bin name**, wobei **name** für die ASM-Datei (ohne .asm) steht.

Nach Abschluss sollte im Verzeichnis eine Datei **name.bin** stehen, die dann zum AVR-Computer via XModem übertragen werden kann.

## 3 Verzeichnis der API-Funktionen

### 3.1 Aufbau der Funktionstabelle

<b>Funktionsname</b>	Name der API-Funktion, entspricht auch dem Makro-Aufruf
<b>Aufruf ohne Makro</b>	Aufruf der Funktion, wenn ohne Makros gearbeitet wird
<b>Funktionsbeschreibung</b>	Kurzbeschreibung der Funktion
<b>Parameter</b>	Registerzuordnung zu den Parametern
<b>Rückgabewerte</b>	Registerzuordnung zu den Rückgabewerten
<b>Register</b>	Register, deren Inhalt verändert wird (ausser Rückgabewerte)
<b>Videomodi</b>	Videomodi, in denen die Funktion sinnvoll einsetzbar ist. Wenn diese Tabellenzeile fehlt, ist die Funktion in allen Videomodi anwendbar.

Zusätzlich zu den angegebenen Registern werden teilweise die Register R0 und R1 sowie das Fehlerregister R24 durch die API-Funktionen überschrieben.

### 3.2 Sichern und Restaurieren von Registern

<b>Funktionsname</b>	<b>api_pushxyz</b>
<b>Aufruf ohne Makro</b>	call api_pushxyz
<b>Funktionsbeschreibung</b>	sichert die Register X,Y und Z auf den Stack
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	R0, R1

<b>Funktionsname</b>	<b>api_popxyz</b>
<b>Aufruf ohne Makro</b>	jmp api_popxyz
<b>Funktionsbeschreibung</b>	holt die Register X,Y und Z vom Stack und führt ein RET aus
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	X, Y, Z

<b>Funktionsname</b>	<b>api_pushregs</b>
<b>Aufruf ohne Makro</b>	call api_pushregs
<b>Funktionsbeschreibung</b>	sichert die Register X,Y und Z sowie R20-R23 auf den Stack
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	R0, R1

<b>Funktionsname</b>	<b>api_popxyz</b>
<b>Aufruf ohne Makro</b>	jmp api_popregs
<b>Funktionsbeschreibung</b>	holt die Register X,Y und Z sowie R20-R23 vom Stack und führt ein RET aus
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	X, Y, Z, R20, R21, R22, R23

### 3.3 Berechnungsfunktionen

Alle Berechnungsfunktionen (außer der Quadratwurzel) beziehen sich auf vorzeichenbehaftete 16 Bit Zahlen. Dabei bildet das X-Register eine Art "Akkumulator", als zweiter Operand wird das Registerpaar R16/R17 genutzt. R24 ist nach der Operation entweder 0 (kein Fehler) oder enthält einen Fehlercode. Die Fehlercodes entsprechen denen im BASIC Programm.

<b>Funktionsname</b>	<b>api_add</b>
<b>Aufruf ohne Makro</b>	call api_add
<b>Funktionsbeschreibung</b>	$X = X + R16/R17$
<b>Parameter</b>	X, R16, R17
<b>Rückgabewerte</b>	X
<b>Register</b>	R18

<b>Funktionsname</b>	<b>api_sub</b>
<b>Aufruf ohne Makro</b>	call api_sub
<b>Funktionsbeschreibung</b>	$X = X - R16/R17$
<b>Parameter</b>	X, R16, R17
<b>Rückgabewerte</b>	X
<b>Register</b>	R16/R17 = -R16/R17

<b>Funktionsname</b>	<b>api_mul</b>
<b>Aufruf ohne Makro</b>	call api_mul
<b>Funktionsbeschreibung</b>	$X = X * R16/R17$
<b>Parameter</b>	X, R16, R17
<b>Rückgabewerte</b>	X
<b>Register</b>	R16...R21

<b>Funktionsname</b>	<b>api_div</b>
<b>Aufruf ohne Makro</b>	call api_div
<b>Funktionsbeschreibung</b>	$X = X / R16/R17$
<b>Parameter</b>	X, R16, R17
<b>Rückgabewerte</b>	X = Quotient, R18/R19 = Rest
<b>Register</b>	R20, R21

<b>Funktionsname</b>	<b>api_sin</b>
<b>Aufruf ohne Makro</b>	call api_sin
<b>Funktionsbeschreibung</b>	$X = 256 * \sin(X)$
<b>Parameter</b>	X (in Grad)
<b>Rückgabewerte</b>	X
<b>Register</b>	Z, R16, R17

<b>Funktionsname</b>	<b>api_cos</b>
<b>Aufruf ohne Makro</b>	call api_cos
<b>Funktionsbeschreibung</b>	$X = 256 * \cos(X)$
<b>Parameter</b>	X (in Grad)
<b>Rückgabewerte</b>	X
<b>Register</b>	Z, R16, R17



<b>Funktionsname</b>	<b>api_eq</b>
<b>Aufruf ohne Makro</b>	call api_eq
<b>Funktionsbeschreibung</b>	gibt 1 zurück, wenn $X == R16/R17$ , ansonsten 0
<b>Parameter</b>	X, R16, R17
<b>Rückgabewerte</b>	X
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_gt</b>
<b>Aufruf ohne Makro</b>	call api_gt
<b>Funktionsbeschreibung</b>	gibt 1 zurück, wenn $X > R16/R17$ , ansonsten 0
<b>Parameter</b>	X, R16, R17
<b>Rückgabewerte</b>	X
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_lt</b>
<b>Aufruf ohne Makro</b>	call api_lt
<b>Funktionsbeschreibung</b>	gibt 1 zurück, wenn $X < R16/R17$ , ansonsten 0
<b>Parameter</b>	X, R16, R17
<b>Rückgabewerte</b>	X
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_sqr</b>
<b>Aufruf ohne Makro</b>	call api_sqr
<b>Funktionsbeschreibung</b>	gibt die Quadratwurzel aus X zurück, X ist in diesem Fall vorzeichenlos
<b>Parameter</b>	X
<b>Rückgabewerte</b>	X
<b>Register</b>	R16, R17, R18, R19

<b>Funktionsname</b>	<b>api_abs</b>
<b>Aufruf ohne Makro</b>	call api_abs
<b>Funktionsbeschreibung</b>	gibt den Absolutwert von X zurück
<b>Parameter</b>	X
<b>Rückgabewerte</b>	X
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_rnd</b>
<b>Aufruf ohne Makro</b>	call api_rnd
<b>Funktionsbeschreibung</b>	gibt eine Zufallszahl zwischen 0 und X-1 zurück
<b>Parameter</b>	X
<b>Rückgabewerte</b>	X
<b>Register</b>	R16...R19

<b>Funktionsname</b>	<b>api_dbit</b>
<b>Aufruf ohne Makro</b>	call api_dbit
<b>Funktionsbeschreibung</b>	berechnet Bitmuster für Zeichensätze im Videomode 4, entspricht der DBIT() Funktion im BASIC
<b>Parameter</b>	X
<b>Rückgabewerte</b>	X
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_adc</b>
<b>Aufruf ohne Makro</b>	call api_adc
<b>Funktionsbeschreibung</b>	startet den ADC und gibt den ADC-Wert zurück.
<b>Parameter</b>	X = Kanal (0...7)
<b>Rückgabewerte</b>	X = ADC Wert
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_scale</b>
<b>Aufruf ohne Makro</b>	call api_scale
<b>Funktionsbeschreibung</b>	Skalierungsfunktion
<b>Parameter</b>	Die Parameter liegen in dem mit <b>api_getpartab</b> definierten Bereich
<b>Rückgabewerte</b>	R16/R17 = Resultat
<b>Register</b>	R4-R7,R18,R19

Die Funktion entspricht dem **SCALE** Befehl im BASIC. Berechnet wird

$$Y = Y0 + ((Y1 - Y0) * (X - X0) / (X1 - X0))$$

Die Parameter liegen hintereinander im Parameterbereich des Interpreters, der durch **api\_getpartab** bestimmt werden kann, wobei immer zuerst das LSB und danach das MSB im Speicher liegt. Die Reihenfolge der Parameter ist dabei wie folgt festgelegt:

<b>Offset</b>	<b>Parameter</b>
<b>0</b>	Y0
<b>2</b>	Y1
<b>4</b>	X0
<b>6</b>	X
<b>8</b>	X1

### 3.4 Programmsteuerung

<b>Funktionsname</b>	<b>api_sync</b>
<b>Aufruf ohne Makro</b>	call api_sync
<b>Funktionsbeschreibung</b>	wartet auf das Ende des gerade dargestellten Halbbildes (unterer Rand des genutzen Bereiches)
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_fast</b>
<b>Aufruf ohne Makro</b>	call api_fast
<b>Funktionsbeschreibung</b>	schaltet die Bildausgabe ab
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_slow</b>
<b>Aufruf ohne Makro</b>	call api_slow
<b>Funktionsbeschreibung</b>	schaltet die Bildausgabe ein
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_wpage</b>
<b>Aufruf ohne Makro</b>	call api_wpage
<b>Funktionsbeschreibung</b>	schreibt eine Flash-Page
<b>Parameter</b>	Y=RAM-Adresse, Z=Flash-Adresse
<b>Rückgabewerte</b>	—
<b>Register</b>	R20...R23,X,Y,Z

### 3.5 Auskunftsfunktionen

<b>Funktionsname</b>	<b>api_version</b>
<b>Aufruf ohne Makro</b>	call api_version
<b>Funktionsbeschreibung</b>	ermittelt die API-Version
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 = API-Version (derzeit 6)
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getvram</b>
<b>Aufruf ohne Makro</b>	call api_getvram
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse des Bildspeichers
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y zeigt auf den Anfang des Bildspeichers
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getsysram</b>
<b>Aufruf ohne Makro</b>	call api_getsysram
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse des System-Speichers
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y zeigt auf den Anfang des System-Speichers
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getpal</b>
<b>Aufruf ohne Makro</b>	call api_getpal
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse des Paletten-Speichers
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y zeigt auf den Anfang des Paletten-Speichers
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getarray</b>
<b>Aufruf ohne Makro</b>	call api_getarray
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse des Arrays
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y zeigt auf den Anfang des Arrays
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getvar</b>
<b>Aufruf ohne Makro</b>	call api_getvar
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse der BASIC-Variablen
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y zeigt auf den Anfang der BASIC-Variablen (Variable "A")
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getprg</b>
<b>Aufruf ohne Makro</b>	call api_getprg
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse der Programmbelegung
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y zeigt auf den Anfang der Belegungstabelle (Word)
<b>Register</b>	—

Die Belegungstabelle besteht aus 8 Werten. Diese sind 8 möglichen Funktionalitäten zugeordnet. Wenn mindestens ein

Programm / Treiber / Bibliothek die Funktionalität anbietet, befindet sich die HIGH-Adresse ider entsprechenden Speicherzelle. Die Adresse bezieht sich auf 16-Bit Werte, so dass der Wert direkt in das ZH-Register für Sprünge und Unterprogrammaufrufe verwendet werden kann. Wird eine Funktionalität nicht angeboten, so befindet sich in der entsprechenden

Speicherzelle der Wert 0	<b>Funktionsname</b>	<b>api_getvalues</b>
	<b>Aufruf ohne Makro</b>	call api_getvalues
	<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse des Funktionsparameter-Blocks
	<b>Parameter</b>	—
	<b>Rückgabewerte</b>	Y zeigt auf den Anfang des Blockes
	<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getpartab</b>
<b>Aufruf ohne Makro</b>	call api_getpartab
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse des (internen) Parameter-Blocks
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y zeigt auf den Anfang des Blockes
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getchart0</b>
<b>Aufruf ohne Makro</b>	call api_getchart0
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse der Zeichentabelle für Font 0 im Flash
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Z zeigt auf den Anfang der Zeichentabelle für Font 0
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getchart1</b>
<b>Aufruf ohne Makro</b>	call api_getchart1
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse der Zeichentabelle für Font 1 im Flash
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Z zeigt auf den Anfang der Zeichentabelle für Font 1
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getbuffer</b>
<b>Aufruf ohne Makro</b>	call api_getbuffer
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse der beiden BASIC Buffer
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y zeigt auf den Anfang der beiden Speicherbereiche
<b>Register</b>	—

- Der BASIC Pufferbereich besteht aus zwei Puffern a 40 Bytes. Der erste Bereich ist der Textpuffer, der z.B. den bei INPUT eingegebenen Text enthält. Außerdem dient er als Source-Puffer bei den **api\_token** und **api\_untoken** Routinen.
- Der zweite Bereich ist der Code-Buffer für das BASIC. Zum einen werden aus diesem Speicherbereich die BASIC-Zeilen interpretiert, zum anderen dient er als Code-Puffer für die **api\_token** und **api\_untoken** Routinen.

<b>Funktionsname</b>	<b>api_getprog</b>
<b>Aufruf ohne Makro</b>	call api_getprog
<b>Funktionsbeschreibung</b>	bestimmt die Nummer des aufrufenden Programmes im Flash
<b>Parameter</b>	—
<b>Rückgabewerte</b>	ZL = Programmnummer (0...7)
<b>Register</b>	ZH

<b>Funktionsname</b>	<b>api_getbase</b>
<b>Aufruf ohne Makro</b>	call api_getbase
<b>Funktionsbeschreibung</b>	bestimmt die Anfangsadresse des aufrufenden Programmes im Flash
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Z zeigt auf den Anfang des aufrufenden Programmes
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_dataptr</b>
<b>Aufruf ohne Makro</b>	call api_dataptr
<b>Funktionsbeschreibung</b>	bestimmt die Adresse eines Datenblocks im aufrufenden Programm
<b>Parameter</b>	X = Offset
<b>Rückgabewerte</b>	Z zeigt auf die absolute Adresse eines Datenblocks im aufrufenden Programm
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_drvcode</b>
<b>Aufruf ohne Makro</b>	call api_drvcode
<b>Funktionsbeschreibung</b>	bestimmt Flagbyte und Drivercode eines geladenen Treibers
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20=Flagbyte, R21=Drivercode
<b>Register</b>	—

### 3.6 Tastatur

<b>Funktionsname</b>	<b>api_waitkey</b>
<b>Aufruf ohne Makro</b>	call api_waitkey
<b>Funktionsbeschreibung</b>	wartet auf einen Tastendruck (ausser SHIFT, CTRL, ALT)
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 = Keycode der gedrückten Taste
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_nokey</b>
<b>Aufruf ohne Makro</b>	call api_nokey
<b>Funktionsbeschreibung</b>	wartet solange, bis keine Taste mehr gedrückt ist (ausser SHIFT, CTRL, ALT)
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_kstate</b>
<b>Aufruf ohne Makro</b>	call api_kstate
<b>Funktionsbeschreibung</b>	gibt den augenblicklichen Zustand der SHIFT-, CTRL- und ALT-Tasten zurück
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 (Bit0=LSHIFT, Bit1=RSHIFT, Bit2=LCTRL, Bit3=RCTRL, Bit4=ALT)
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_keycode</b>
<b>Aufruf ohne Makro</b>	call api_keycode
<b>Funktionsbeschreibung</b>	gibt den Keycode der gerade gedrückten Taste oder 0x00 (keine Taste gedrückt) zurück
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 = Keycode
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_lastkey</b>
<b>Aufruf ohne Makro</b>	call api_lastkey
<b>Funktionsbeschreibung</b>	gibt den Keycode der zuletzt gedrückten Taste oder 0x00 (noch keine Taste gedrückt) zurück
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 = Keycode
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_scancode</b>
<b>Aufruf ohne Makro</b>	call api_scancode
<b>Funktionsbeschreibung</b>	gibt den Scancode der zuletzt gedrückten Taste zurück
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 = Scancode
<b>Register</b>	—

### 3.7 Bildschirmausgabe

<b>Funktionsname</b>	<b>api_clrscr</b>
<b>Aufruf ohne Makro</b>	call api_clrscr
<b>Funktionsbeschreibung</b>	löscht den Bildschirm mit der aktuellen Farbeinstellung
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	R20
<b>Videomodi</b>	0...6

<b>Funktionsname</b>	<b>api_clearvec</b>
<b>Aufruf ohne Makro</b>	call api_clearvec
<b>Funktionsbeschreibung</b>	löscht den Vektorbereich im Grafikmode 7
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	7

<b>Funktionsname</b>	<b>api_cleartext</b>
<b>Aufruf ohne Makro</b>	call api_cleartext
<b>Funktionsbeschreibung</b>	löscht den Textbereich im Grafikmode 7 mit der aktuellen Farbeinstellung
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	7

<b>Funktionsname</b>	<b>api_gotoxy</b>
<b>Aufruf ohne Makro</b>	call api_gotoxy
<b>Funktionsbeschreibung</b>	setzt die Position des Text-Cursors (auch in den Grafikmodi)
<b>Parameter</b>	XL = X-Position, XH = Y-Position, In den Grafikmodi als Pixelposition
<b>Rückgabewerte</b>	—
<b>Register</b>	Im Textmodus werden XL und XH auf die maximalen Bildschirmkoordinaten begrenzt
<b>Videomodi</b>	alle

Bei den Ausgabefunktionen bestimmt R25 den Modus, die Werte sind beim PRINT Befehl näher erläutert. Ebenso bestimmt der eingestellte Channel, wohin ausgegeben wird.

<b>Funktionsname</b>	<b>api_outchar</b>
<b>Aufruf ohne Makro</b>	call api_outchar
<b>Funktionsbeschreibung</b>	gibt ein Zeichen aus
<b>Parameter</b>	R20 = Zeichen, R25 = Mode
<b>Rückgabewerte</b>	—
<b>Register</b>	R20, R21
<b>Videomodi</b>	alle

<b>Funktionsname</b>	<b>api_newline</b>
<b>Aufruf ohne Makro</b>	call api_newline
<b>Funktionsbeschreibung</b>	gibt einen Zeilenvorschub aus
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	R20, R21
<b>Videomodi</b>	0, 4, 6



<b>Funktionsname</b>	<b>api_outdez</b>
<b>Aufruf ohne Makro</b>	call api_outdez
<b>Funktionsbeschreibung</b>	gibt den Inhalt von X dezimal aus (-32768...32767)
<b>Parameter</b>	X= 16-Bit Wert, R25 = Mode/Format
<b>Rückgabewerte</b>	—
<b>Register</b>	R20, R21
<b>Videomodi</b>	alle

<b>Funktionsname</b>	<b>api_outhex</b>
<b>Aufruf ohne Makro</b>	call api_outhex
<b>Funktionsbeschreibung</b>	gibt den Inhalt von X hexadezimal aus (00...FF oder 0000...FFFF)
<b>Parameter</b>	X= 16-Bit Wert, R25 = Mode/Format
<b>Rückgabewerte</b>	—
<b>Register</b>	R20, R21
<b>Videomodi</b>	alle

<b>Funktionsname</b>	<b>api_romtext</b>
<b>Aufruf ohne Makro</b>	call api_romtext
<b>Funktionsbeschreibung</b>	gibt einen nullterminierten String aus dem Flash aus
<b>Parameter</b>	Z = Zeiger auf den String im Flash
<b>Rückgabewerte</b>	—
<b>Register</b>	Z, R20, R21
<b>Videomodi</b>	alle

<b>Funktionsname</b>	<b>api_thistext</b>
<b>Aufruf ohne Makro</b>	call api_thistext
<b>Funktionsbeschreibung</b>	gibt einen nullterminierten String aus dem Flash aus, der direkt auf den Aufruf folgt.
<b>Parameter</b>	Wenn das erste Byte 0xFF ist, folgt direkt der auszugebende String, andernfalls wird das erste Byte als Y-Koordinate und das zweite als X-Koordinate gewertet, bevor die eigentliche Zeichenkette folgt.
<b>Rückgabewerte</b>	—
<b>Register</b>	Z, R20, R21
<b>Videomodi</b>	alle

<b>Funktionsname</b>	<b>api_cbox</b>
<b>Aufruf ohne Makro</b>	call api_cbox
<b>Funktionsbeschreibung</b>	Löscht eine Rechteckfläche mit der eingestellten Farbe
<b>Parameter</b>	YL = X1, YH = Y1, ZL = X2, ZH = Y2
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 1, 4

<b>Funktionsname</b>	<b>api_ibox</b>
<b>Aufruf ohne Makro</b>	call api_ibox
<b>Funktionsbeschreibung</b>	Invertiert eine Rechteckfläche (tauscht Vorder- mit Hintergrundfarbe)
<b>Parameter</b>	YL = X1, YH = Y1, ZL = X2, ZH = Y2
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 1, 4

<b>Funktionsname</b>	<b>api_scroll</b>
<b>Aufruf ohne Makro</b>	call api_scroll
<b>Funktionsbeschreibung</b>	Scrollt den inneren Bildschirmbereich, siehe SCROLL Befehl
<b>Parameter</b>	R20 = Richtung (0=nach oben, 1=nach rechts, 2=nach unten, 3=nach links)
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 4, 6

<b>Funktionsname</b>	<b>api_sprite</b>
<b>Aufruf ohne Makro</b>	call api_sprite
<b>Funktionsbeschreibung</b>	stellt ein Sprite dar (siehe BASIC Refrenz)
<b>Parameter</b>	XL=X, XH=Y, Y=Pointer auf Sprite-Daten
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 4, 6

<b>Funktionsname</b>	<b>api_copypchar4</b>
<b>Aufruf ohne Makro</b>	call api_copypchar4
<b>Funktionsbeschreibung</b>	kopiert ein Zeichen aus dem System-Zeichensatz in den User-Zeichensatz für Videomode 4
<b>Parameter</b>	R20 = System-Zeichen, R21 = User Zeichen (0...127)
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	4

<b>Funktionsname</b>	<b>api_copypchar6</b>
<b>Aufruf ohne Makro</b>	call api_copypchar6
<b>Funktionsbeschreibung</b>	kopiert ein Zeichen aus dem System-Zeichensatz in den User-Zeichensatz für Videomode 6
<b>Parameter</b>	R20 = System-Zeichen, R21 = User Zeichen (0...63), R22 = VG/HG Farbe
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	6

<b>Funktionsname</b>	<b>api_sbackup</b>
<b>Aufruf ohne Makro</b>	call api_sbackup
<b>Funktionsbeschreibung</b>	kopiert den sichtbaren Screen-Bereich in den Monitor-Backup Screen
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	R20, X, Y, Z
<b>Videomodi</b>	nur bei 1 und 5 sinnvoll

<b>Funktionsname</b>	<b>api_srestore</b>
<b>Aufruf ohne Makro</b>	call api_srestore
<b>Funktionsbeschreibung</b>	kopiert den Monitor-Backup Screen in den sichtbaren Screen-Bereich
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	R20, X, Y, Z
<b>Videomodi</b>	nur bei 1 und 5 sinnvoll

### 3.8 Grafik

<b>Funktionsname</b>	<b>api_plot</b>
<b>Aufruf ohne Makro</b>	call api_plot
<b>Funktionsbeschreibung</b>	Zeichnet einen Punkt in der eingestellten Vordergrundfarbe
<b>Parameter</b>	XL=X, XH=Y
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 1, 2, 3, (4), 5

<b>Funktionsname</b>	<b>api_draw</b>
<b>Aufruf ohne Makro</b>	call api_draw
<b>Funktionsbeschreibung</b>	Zeichnet eine Linie in der eingestellten Vordergrundfarbe
<b>Parameter</b>	YL=X1, YH=Y1, ZL=X2, ZH=Y2
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 1, 2, 3, (4), 5

<b>Funktionsname</b>	<b>api_box</b>
<b>Aufruf ohne Makro</b>	call api_box
<b>Funktionsbeschreibung</b>	Zeichnet ein Rechteck in der eingestellten Vordergrundfarbe
<b>Parameter</b>	YL=X1, YH=Y1, ZL=X2, ZH=Y2
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 1, 2, 3, (4), 5

<b>Funktionsname</b>	<b>api_fbox</b>
<b>Aufruf ohne Makro</b>	call api_fbox
<b>Funktionsbeschreibung</b>	Zeichnet ein ausgefülltes Rechteck in der eingestellten Vordergrundfarbe
<b>Parameter</b>	YL=X1, YH=Y1, ZL=X2, ZH=Y2
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 1, 2, 3, (4), 5

<b>Funktionsname</b>	<b>api_circle</b>
<b>Aufruf ohne Makro</b>	call api_circle
<b>Funktionsbeschreibung</b>	Zeichnet einen Kreis/Ellipse in der eingestellten Vordergrundfarbe
<b>Parameter</b>	YL=X, YH=Y, ZL=RX, ZH=RY
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 1, 2, 3, (4), 5

<b>Funktionsname</b>	<b>api_fcircle</b>
<b>Aufruf ohne Makro</b>	call api_fcircle
<b>Funktionsbeschreibung</b>	Zeichnet einen ausgefülltesn Kreis in der eingestellten Vordergrundfarbe
<b>Parameter</b>	YL=X, YH=Y, ZL=RX, ZH=RY
<b>Rückgabewerte</b>	—
<b>Register</b>	—
<b>Videomodi</b>	0, 1, 2, 3, (4), 5

<b>Funktionsname</b>	<b>api_bcopy1</b>
<b>Aufruf ohne Makro</b>	call api_bcopy1
<b>Funktionsbeschreibung</b>	Kopiert einen Bildschirmausschnitt an eine andere Stelle
<b>Parameter</b>	YL=SRCX, YH=SRCY, ZL=DESTX, ZH=DESTY, R21=Bytes/Zeile, R22=Zeilen
<b>Rückgabewerte</b>	—
<b>Register</b>	R4ldotsR7
<b>Videomodi</b>	1, 2, 3, 5

<b>Funktionsname</b>	<b>api_bcopy2</b>
<b>Aufruf ohne Makro</b>	call api_bcopy2
<b>Funktionsbeschreibung</b>	Kopiert einen Bildschirmausschnitt vom Bildspeicher in das RAM
<b>Parameter</b>	YL=SRCX, YH=SRCY, Z=RAM-Adresse, R21=Bytes/Zeile, R22=Zeilen
<b>Rückgabewerte</b>	—
<b>Register</b>	R4ldotsR7
<b>Videomodi</b>	1, 2, 3, 5

<b>Funktionsname</b>	<b>api_bcopy3</b>
<b>Aufruf ohne Makro</b>	call api_bcopy3
<b>Funktionsbeschreibung</b>	Kopiert einen Bildschirmausschnitt RAM in den Bildspeicher
<b>Parameter</b>	YL=DESTX, YH=DESTY, Z=RAM-Adresse
<b>Rückgabewerte</b>	—
<b>Register</b>	R4ldotsR7
<b>Videomodi</b>	1, 2, 3, 5

### 3.9 Kommunikation

<b>Funktionsname</b>	<b>api_putpar</b>
<b>Aufruf ohne Makro</b>	call api_putpar
<b>Funktionsbeschreibung</b>	gibt ein Zeichen über die parallele Schnittstelle aus
<b>Parameter</b>	R20 = Zeichen
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getser</b>
<b>Aufruf ohne Makro</b>	call api_getser
<b>Funktionsbeschreibung</b>	wartet auf ein Zeichen von der seriellen Schnittstelle, keine Abbruchmöglichkeit
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 = Zeichen
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getserb</b>
<b>Aufruf ohne Makro</b>	call api_getserb
<b>Funktionsbeschreibung</b>	wartet auf ein Zeichen von der seriellen Schnittstelle, Abbruch mit ESC möglich
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 = Zeichen oder 0x1B (bei Abbruch mit ESC)
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_putser</b>
<b>Aufruf ohne Makro</b>	call api_putser
<b>Funktionsbeschreibung</b>	gibt ein Zeichen über die serielle Schnittstelle aus
<b>Parameter</b>	R20 = Zeichen
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_putsernl</b>
<b>Aufruf ohne Makro</b>	call api_putsernl
<b>Funktionsbeschreibung</b>	gibt einen Zeilenvorschub über die serielle Schnittstelle aus
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	Y = Y + 128

<b>Funktionsname</b>	<b>api_hexline</b>
<b>Aufruf ohne Makro</b>	call api_hexline
<b>Funktionsbeschreibung</b>	gibt einen Speicherbereich als Hexstring mit abschliessendem Zeilenvorschub über die serielle Schnittstelle aus
<b>Parameter</b>	Y = Adresse RAM Block, ZL=Anzahl Bytes
<b>Rückgabewerte</b>	—
<b>Register</b>	ZL, R20, Y = Y + 128

<b>Funktionsname</b>	<b>api_putxm</b>
<b>Aufruf ohne Makro</b>	call api_putxm
<b>Funktionsbeschreibung</b>	gibt einen X-Modem Block über die serielle Schnittstelle aus
<b>Parameter</b>	R22 = Blocknummer, Y = Adresse RAM Block
<b>Rückgabewerte</b>	—
<b>Register</b>	Y = Y + 128

<b>Funktionsname</b>	<b>api_getxm</b>
<b>Aufruf ohne Makro</b>	call api_getxm
<b>Funktionsbeschreibung</b>	liest einen X-Modem Block über die serielle Schnittstelle ein
<b>Parameter</b>	Y = Adresse RAM Block
<b>Rückgabewerte</b>	R22 = Blocknummer
<b>Register</b>	Y = Y + 128

<b>Funktionsname</b>	<b>api_preceive</b>
<b>Aufruf ohne Makro</b>	call api_preceive
<b>Funktionsbeschreibung</b>	lädt ein Programm (im Binärmode) über das X-Modem Protokoll
<b>Parameter</b>	R15 = Programmnummer-1 (0...7]
<b>Rückgabewerte</b>	
<b>Register</b>	R20...R23,X,Y,Z

<b>Funktionsname</b>	<b>api_psend</b>
<b>Aufruf ohne Makro</b>	call api_psend
<b>Funktionsbeschreibung</b>	sendet ein Programm (im Binärmode) über das X-Modem Protokoll
<b>Parameter</b>	R15 = Programmnummer-1 (0...7]
<b>Rückgabewerte</b>	
<b>Register</b>	R20...R23,X,Y,Z

### 3.10 Audio

<b>Funktionsname</b>	<b>api_note</b>
<b>Aufruf ohne Makro</b>	call api_note
<b>Funktionsbeschreibung</b>	Spielt eine Note mit der aktuellen Lautstärke
<b>Parameter</b>	XL=Notenwert, entsprechend dem BASIC Befehl <b>note</b>
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_setvolume</b>
<b>Aufruf ohne Makro</b>	call api_setvolume
<b>Funktionsbeschreibung</b>	setzt die globale Lautstärke
<b>Parameter</b>	XL=Lautstärkewert (0...15)
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_startseq</b>
<b>Aufruf ohne Makro</b>	call api_startseq
<b>Funktionsbeschreibung</b>	startet den Sequenzer
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Y=Startadresse (RAM), Z=Endadresse+1 (RAM), XL=Geschwindigkeit, XH=Zyklusanzahl
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_setseq</b>
<b>Aufruf ohne Makro</b>	call api_setseq
<b>Funktionsbeschreibung</b>	setzt den Zyklenzaehler des Sequenzers
<b>Parameter</b>	XL = neuer Zyklus-Wert
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_getseq</b>
<b>Aufruf ohne Makro</b>	call api_getseq
<b>Funktionsbeschreibung</b>	liest den Zyklenzaehler des Sequenzers
<b>Parameter</b>	XL = aktueller Zyklus-Wert
<b>Rückgabewerte</b>	—
<b>Register</b>	—

### 3.11 Zugriff auf das interne EEPROM

Die Funktionen für das interne EEPROM funktionieren nur mit den Speicherzellen 0...2047, befindet sich die Adresse (Y-Register) außerhalb dieses Bereiches, wird das Fehlerregister (R24) auf 34 gesetzt. Da die Speicherzellen ab Adresse 2040 für Systemeinstellungen genutzt werden, sollten diese nach Möglichkeit nicht überschrieben werden.

Vor jedem Schreibvorgang wird die EEPROM-Zelle gelesen, falls Soll- und Istzustand bereits übereinstimmen wird der Schreibvorgang abgebrochen.

<b>Funktionsname</b>	<b>api_eep_read</b>
<b>Aufruf ohne Makro</b>	call api_eep_read
<b>Funktionsbeschreibung</b>	liest ein Byte aus dem internen EEPROM
<b>Parameter</b>	Y=Adresse
<b>Rückgabewerte</b>	R20=Byte
<b>Register</b>	Y=Y+1

<b>Funktionsname</b>	<b>api_eep_write</b>
<b>Aufruf ohne Makro</b>	call api_eep_write
<b>Funktionsbeschreibung</b>	schreibt ein Byte in das interne EEPROM
<b>Parameter</b>	Y=Adresse, R20=Byte
<b>Rückgabewerte</b>	—
<b>Register</b>	Y=Y+1

### 3.12 Funktionen für den I2C-Bus

Der I2C-Bus kann über die API-Funktionen nur als Master genutzt werden.

<b>Funktionsname</b>	<b>api_i2c_start</b>
<b>Aufruf ohne Makro</b>	call api_i2c_start
<b>Funktionsbeschreibung</b>	erzeugt eine Start-Kondition auf dem I2C-Bus
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_i2c_stop</b>
<b>Aufruf ohne Makro</b>	call api_i2c_stop
<b>Funktionsbeschreibung</b>	erzeugt eine Stop-Kondition auf dem I2C-Bus
<b>Parameter</b>	—
<b>Rückgabewerte</b>	—
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_i2c_rbyte</b>
<b>Aufruf ohne Makro</b>	call api_i2c_rbyte
<b>Funktionsbeschreibung</b>	liest ein Byte vom I2C-Bus ein und sendet ein ACK
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R21=Byte, R22=I2C Status-Register
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_i2c_rbyten</b>
<b>Aufruf ohne Makro</b>	call api_i2c_rbyten
<b>Funktionsbeschreibung</b>	liest ein Byte vom I2C-Bus ein ohne ein ACK zu senden
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R21=Byte, R22=I2C Status-Register
<b>Register</b>	R20



<b>Funktionsname</b>	<b>api_i2c_wbyte</b>
<b>Aufruf ohne Makro</b>	call api_i2c_wbyte
<b>Funktionsbeschreibung</b>	sendet ein Byte auf den I2C-Bus und wartet auf ACK/NAK
<b>Parameter</b>	R21=Byte
<b>Rückgabewerte</b>	R22=I2C Status-Register
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_i2c_read</b>
<b>Aufruf ohne Makro</b>	call api_i2c_read
<b>Funktionsbeschreibung</b>	liest ein Datenbyte von einem externen EEPROM ein
<b>Parameter</b>	X=Speicheradresse im EEPROM, R23=EEPROM Adresse (0...7)
<b>Rückgabewerte</b>	R21=Byte R24=Fehlerstatus
<b>Register</b>	X=X+1, R20, R22

<b>Funktionsname</b>	<b>api_i2c_write</b>
<b>Aufruf ohne Makro</b>	call api_i2c_write
<b>Funktionsbeschreibung</b>	schreibt ein Datenbyte in ein externes EEPROM
<b>Parameter</b>	R21=Byte X=Speicheradresse im EEPROM, R23=EEPROM Adresse (0...7)
<b>Rückgabewerte</b>	R24=Fehlerstatus
<b>Register</b>	X=X+1, R20, R22

<b>Funktionsname</b>	<b>api_i2c_rlm75</b>
<b>Aufruf ohne Makro</b>	call api_i2c_rlm75
<b>Funktionsbeschreibung</b>	liest den Temperaturwert von einem angeschlossenen LM75 Temperatursensor ein
<b>Parameter</b>	R23=LM75 Adresse (0...7)
<b>Rückgabewerte</b>	R24=Fehlerstatus, X=Temperaturwert in 1/2 Grad
<b>Register</b>	R20, R21, R22

### 3.13 Alert- und Abfrageboxen

<b>Funktionsname</b>	<b>api_alert</b>
<b>Aufruf ohne Makro</b>	call api_alert
<b>Funktionsbeschreibung</b>	gibt eine Alertbox aus, nach ENTER wird der Bildschirm restauriert
<b>Parameter</b>	Y zeigt auf Farbbyte (FG/HG) und nullterminierten Text im RAM
<b>Rückgabewerte</b>	—
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_alert_nb</b>
<b>Aufruf ohne Makro</b>	call api_alert_nb
<b>Funktionsbeschreibung</b>	gibt eine Alertbox aus, nach ENTER wird der Bildschirm NICHT restauriert
<b>Parameter</b>	Y zeigt auf Farbbyte (FG/HG) und nullterminierten Text im RAM
<b>Rückgabewerte</b>	—
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_alertthis</b>
<b>Aufruf ohne Makro</b>	call api_alertthis
<b>Funktionsbeschreibung</b>	gibt eine Alertbox aus, nach ENTER wird der Bildschirm restauriert
<b>Parameter</b>	das Farbbyte und der Text liegen nullterminiert nach dem Aufruf
<b>Rückgabewerte</b>	—
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_alertthis_nb</b>
<b>Aufruf ohne Makro</b>	call api_alertthis_nb
<b>Funktionsbeschreibung</b>	gibt eine Alertbox aus, nach ENTER wird der Bildschirm NICHT restauriert
<b>Parameter</b>	das Farbbyte und der Text liegen nullterminiert nach dem Aufruf
<b>Rückgabewerte</b>	—
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_ask</b>
<b>Aufruf ohne Makro</b>	call api_ask
<b>Funktionsbeschreibung</b>	gibt eine Fragebox aus, nach Y/N/ENTER/ESC wird der Bildschirm restauriert
<b>Parameter</b>	Y zeigt auf Farbbyte (FG/HG) und nullterminierten Text im RAM
<b>Rückgabewerte</b>	T-Flag ist gesetzt bei Y/ENTER
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_ask_nb</b>
<b>Aufruf ohne Makro</b>	call api_ask_nb
<b>Funktionsbeschreibung</b>	gibt eine Fragebox aus, nach Y/N/ENTER/ESC wird der Bildschirm NICHT restauriert
<b>Parameter</b>	Y zeigt auf Farbbyte (FG/HG) und nullterminierten Text im RAM
<b>Rückgabewerte</b>	T-Flag ist gesetzt bei Y/ENTER
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_askthis</b>
<b>Aufruf ohne Makro</b>	call api_askthis
<b>Funktionsbeschreibung</b>	gibt eine Fragebox aus, nach Y/N/ENTER/ESC wird der Bildschirm restauriert
<b>Parameter</b>	das Farbbyte und der Text liegen nullterminiert nach dem Aufruf
<b>Rückgabewerte</b>	T-Flag ist gesetzt bei Y/ENTER
<b>Register</b>	R20

<b>Funktionsname</b>	<b>api_askthis_nb</b>
<b>Aufruf ohne Makro</b>	call api_askthis_nb
<b>Funktionsbeschreibung</b>	gibt eine Fragebox aus, nach Y/N/ENTER/ESC wird der Bildschirm NICHT restauriert
<b>Parameter</b>	das Farbbyte und der Text liegen nullterminiert nach dem Aufruf
<b>Rückgabewerte</b>	T-Flag ist gesetzt bei Y/ENTER
<b>Register</b>	R20

### 3.14 Zugriff auf das Dataflash-Dateisystem

<b>Funktionsname</b>	<b>api_fs_size</b>
<b>Aufruf ohne Makro</b>	call api_fs_size
<b>Funktionsbeschreibung</b>	bestimmt die Größe des angeschlossenen Dataflash-Moduls
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20=(Anzahl der Sektoren / 256) oder 0
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_fs_create</b>
<b>Aufruf ohne Makro</b>	call api_fs_create
<b>Funktionsbeschreibung</b>	erzeugt eine Datei
<b>Parameter</b>	R20=Dateinummer, R21=Anzahl Sektoren, R22=Dateityp
<b>Rückgabewerte</b>	R24 (Fehlercode oder 0)
<b>Register</b>	Array(512-767)

<b>Funktionsname</b>	<b>api_fs_delete</b>
<b>Aufruf ohne Makro</b>	call api_fs_delete
<b>Funktionsbeschreibung</b>	löscht eine Datei
<b>Parameter</b>	R20=Dateinummer
<b>Rückgabewerte</b>	R24 (Fehlercode oder 0)
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_fs_read</b>
<b>Aufruf ohne Makro</b>	call api_fs_read
<b>Funktionsbeschreibung</b>	liest einen Dateisektor in einen Bereich des Arrays ein
<b>Parameter</b>	R20=Dateinummer, R21=Sektornummer, R22=Array-Drittel (0,1,2)
<b>Rückgabewerte</b>	R24 (Fehlercode oder 0)
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_fs_write</b>
<b>Aufruf ohne Makro</b>	call api_fs_write
<b>Funktionsbeschreibung</b>	schreibt einen Dateisektor aus einen Bereich des Arrays
<b>Parameter</b>	R20=Dateinummer, R21=Sektornummer, R22=Array-Drittel (0,1,2)
<b>Rückgabewerte</b>	R24 (Fehlercode oder 0)
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_fs_cfree</b>
<b>Aufruf ohne Makro</b>	call api_fs_cfree
<b>Funktionsbeschreibung</b>	Zählt die freien Programmplätze und Datensektoren
<b>Parameter</b>	—
<b>Rückgabewerte</b>	Z=freie Programmplätze, Y=freie Datensektoren
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_fs_checkf</b>
<b>Aufruf ohne Makro</b>	call api_fs_checkf
<b>Funktionsbeschreibung</b>	ermittelt, ob das Dataflash-Modul formatiert ist
<b>Parameter</b>	—
<b>Rückgabewerte</b>	R20 (1=formatiert, sonst 0)
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_fs_gettype</b>
<b>Aufruf ohne Makro</b>	call api_fs_gettype
<b>Funktionsbeschreibung</b>	bestimmt den Dateityp einer Datei
<b>Parameter</b>	R20=Dateinummer
<b>Rückgabewerte</b>	R20=Dateityp-Code
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_fs_fsize</b>
<b>Aufruf ohne Makro</b>	call api_fs_fsize
<b>Funktionsbeschreibung</b>	bestimmt die Größe einer Datei (in Pages) oder den freien Speicher auf dem Dataflash Modul
<b>Parameter</b>	X=Dateinummer oder -1 (freie Dateien) oder -2 (freie Pages)
<b>Rückgabewerte</b>	X=Anzahl der Pages/Dateien
<b>Register</b>	Y, Z

<b>Funktionsname</b>	<b>api_fs_ffind</b>
<b>Aufruf ohne Makro</b>	call api_fs_ffind
<b>Funktionsbeschreibung</b>	sucht nach einer Datei
<b>Parameter</b>	X=Arrayposition (siehe FFIND Funktion)
<b>Rückgabewerte</b>	X=Dateinummer oder -1
<b>Register</b>	Y, Z

<b>Funktionsname</b>	<b>api_fs_check</b>
<b>Aufruf ohne Makro</b>	call api_fs_check
<b>Funktionsbeschreibung</b>	testet das Dataflash auf gültiges Dateisystem
<b>Parameter</b>	
<b>Rückgabewerte</b>	
<b>Register</b>	R20

Diese Funktion dient zum Vorbereiten der Fileselectorbox und setzt die notwendigen internen Variablen. Tritt ein Fehler auf, dann wird die Rücksprungadresse vom Stack entfernt und nach Anzeige einer entsprechenden Alertbox zu der Adresse des vorherigen Aufrufes zurückgesprungen. Aus diesem Grunde ist es sinnvoll, z.B. Lade- und Speicherfunktionen samt der Fileselectorbox-Aufrufe in eine eigene Subroutine zu packen. Bei Fehlern wird diese dann einfach verlassen und in das übergeordnete Programm zurückgesprungen.

<b>Funktionsname</b>	<b>api_fs_fsel</b>
<b>Aufruf ohne Makro</b>	call api_fs_fsel
<b>Funktionsbeschreibung</b>	ruft die Fileselectorbox auf (nur im Videomode 0)
<b>Parameter</b>	der Kopftext liegt nullterminiert nach dem Aufruf
<b>Rückgabewerte</b>	R23=Dateinummer, R20=Dateityp
<b>Register</b>	—

Wird die Dateiauswahl mittels **ESC** Taste abgebrochen, dann ist der in R20 zurückgegebene Dateityp 0xed.

<b>Funktionsname</b>	<b>api_fs_fsel_nb</b>
<b>Aufruf ohne Makro</b>	call api_fs_fsel_nb
<b>Funktionsbeschreibung</b>	ruft die Fileselectorbox ohne Screen-Backup auf (nur im Videomode 0)
<b>Parameter</b>	der Kopftext liegt nullterminiert nach dem Aufruf
<b>Rückgabewerte</b>	R23=Dateinummer, R20=Dateityp
<b>Register</b>	—

Wird die Dateiauswahl mittels **ESC** Taste abgebrochen, dann ist der in R20 zurückgegebene Dateityp 0xed.

### 3.15 Direktzugriff auf das Dataflash

<b>Funktionsname</b>	<b>api_fs_rread</b>
<b>Aufruf ohne Makro</b>	call api_fs_rread
<b>Funktionsbeschreibung</b>	liest einen Sektor aus dem Dataflash
<b>Parameter</b>	X=Sektor-Nummer, Y=Adresse im RAM
<b>Rückgabewerte</b>	R24 (Fehlercode oder 0)
<b>Register</b>	Y=Y+256

<b>Funktionsname</b>	<b>api_fs_rwrite</b>
<b>Aufruf ohne Makro</b>	call api_fs_rwritw
<b>Funktionsbeschreibung</b>	schreibt einen Sektor in das Dataflash
<b>Parameter</b>	X=Sektor-Nummer, Y=Adresse im RAM
<b>Rückgabewerte</b>	R24 (Fehlercode oder 0)
<b>Register</b>	Y=Y+256

### 3.16 BASIC/SYSTEM-Funktionen

<b>Funktionsname</b>	<b>api_lfind</b>
<b>Aufruf ohne Makro</b>	call api_lfind
<b>Funktionsbeschreibung</b>	sucht im System nach einer Bibliothek
<b>Parameter</b>	X = Bibliothek code
<b>Rückgabewerte</b>	R21 = Programmplatz (1...8) oder 0 für nicht gefunden
<b>Register</b>	Z, R22-R23

<b>Funktionsname</b>	<b>api_lcall</b>
<b>Aufruf ohne Makro</b>	call api_lcall
<b>Funktionsbeschreibung</b>	Ruft eine Funktion in einer Library auf
<b>Parameter</b>	R20 = Bibliothek(1...8), R21 = Funktionsnummer
<b>Rückgabewerte</b>	evtl. von aufgerufener Funktion
<b>Register</b>	Z, R0, R1 und Register des Funktionsaufrufes

<b>Funktionsname</b>	<b>api_token</b>
<b>Aufruf ohne Makro</b>	call api_token
<b>Funktionsbeschreibung</b>	tokenisiert eine Zeile aus dem Source-Puffer in den Code-Puffer
<b>Parameter</b>	Quelltext im Source-Puffer
<b>Rückgabewerte</b>	tokenisierte Zeile im Code-Puffer
<b>Register</b>	X, Y, Z, R20-R23

<b>Funktionsname</b>	<b>api_untoken</b>
<b>Aufruf ohne Makro</b>	call api_untoken
<b>Funktionsbeschreibung</b>	übersetzt eine tokenisierte Zeile zurück in den Sourcecode
<b>Parameter</b>	tokenisierte Zeile im Code-Puffer
<b>Rückgabewerte</b>	Quelltext im Source-Puffer
<b>Register</b>	X, Y, Z, R20...R23

<b>Funktionsname</b>	<b>api_basrun</b>
<b>Aufruf ohne Makro</b>	call api_basrun
<b>Funktionsbeschreibung</b>	führt die im Code-Puffer befindliche Zeile aus
<b>Parameter</b>	R12=Zeilennummer, R13=erstes Statement, Zeile im Code-Puffer
<b>Rückgabewerte</b>	—
<b>Register</b>	X, Y, Z, R4...R7, R20...R23

<b>Funktionsname</b>	<b>api_expaser</b>
<b>Aufruf ohne Makro</b>	call api_expaser
<b>Funktionsbeschreibung</b>	interpretiert eine Formel im Textformat
<b>Parameter</b>	Z = Zeiger auf Funktionstext
<b>Rückgabewerte</b>	X = Resultat (16 Bit signed)
<b>Register</b>	Y, Z, R4...R7, R20...R23

<b>Funktionsname</b>	<b>api_tparser</b>
<b>Aufruf ohne Makro</b>	call api_tparser
<b>Funktionsbeschreibung</b>	stellt einen Tokenparser zur Verfügung
<b>Parameter</b>	Z = Zeiger auf Tokentabelle im Flash, Y = Zeiger auf Text im RAM, R22 = Länge eines Eintrages in Bytes
<b>Rückgabewerte</b>	R20=gefundener Token oder 0
<b>Register</b>	Y steht hinter dem gefundenen Text

In der Tokentabelle stehen immer zuerst der Tokenwert und danach der Tokentext, wobei mit Punkten aufgefüllt wird. Als letztes sollte ein Eintrag nur aus Punkten stehen, dieser fängt dann ungültige Worte ab. Die Funktion kann auch mehrmals nacheinander aufgerufen werden, als Trennzeichen können der Leerraum und das Komma verwendet werden. Im folgenden Beispiel wäre die Länge eines Eintrages 6 Bytes (1 Byte Tokenwert und 5 Bytes Text):

```
tokentab:
.db 0x01, "LOAD."
.db 0x02, "ADD.."
.db 0x03, "SUB.."
.db 0x00, "....."
```

<b>Funktionsname</b>	<b>api_arrview</b>
<b>Aufruf ohne Makro</b>	call api_arrview
<b>Funktionsbeschreibung</b>	Array-Ansicht
<b>Parameter</b>	R23 = Anzeigeposition (*64)
<b>Rückgabewerte</b>	—
<b>Register</b>	—

Diese Funktion ist nur in Videomodus 0 sinnvoll.



### 3.17 Nutzung des Videomode 0

Die folgenden Funktionen dienen dazu, Programm-Routinen des Videomode 0 in eigenen Treibern zu verwenden.

<b>Funktionsname</b>	<b>api_vm0cls</b>
<b>Aufruf ohne Makro</b>	jmp api_vm0cls
<b>Funktionsbeschreibung</b>	springt zur CLS-Routine von Videomode 0
<b>Parameter</b>	nur in Treibern nutzbar
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_vm0char</b>
<b>Aufruf ohne Makro</b>	jmp api_vm0char
<b>Funktionsbeschreibung</b>	springt zur Zeichenausgabe-Routine von Videomode 0
<b>Parameter</b>	nur in Treibern nutzbar
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_vm0gotoxy</b>
<b>Aufruf ohne Makro</b>	jmp api_vm0gotoxy
<b>Funktionsbeschreibung</b>	springt zur Positionierungs-Routine von Videomode 0
<b>Parameter</b>	nur in Treibern nutzbar
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_vm0plot</b>
<b>Aufruf ohne Makro</b>	jmp api_vm0plot
<b>Funktionsbeschreibung</b>	springt zur Pixelsetz-Routine von Videomode 0
<b>Parameter</b>	nur in Treibern nutzbar
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_vm0line</b>
<b>Aufruf ohne Makro</b>	jmp api_vm0line
<b>Funktionsbeschreibung</b>	gibt eine Pixelzeile im Videomode 0 aus
<b>Parameter</b>	nur in Treibern nutzbar
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_vm0newline</b>
<b>Aufruf ohne Makro</b>	jmp api_vm0newline
<b>Funktionsbeschreibung</b>	gibt einen Zeilenvorschub im Videomode 0 aus.
<b>Parameter</b>	nur in Treibern nutzbar
<b>Rückgabewerte</b>	—
<b>Register</b>	—

### 3.18 Array-Funktionen

Diese Funktionen können sowohl das interne Array als auch externen Speicher nutzen. Lässt sich die entsprechende Arrayzelle nicht ansprechen, wird der Fehler 18 (OUT OF ARRAY) im Fehlerregister zurückgegeben.

<b>Funktionsname</b>	<b>api_aread</b>
<b>Aufruf ohne Makro</b>	call api_aread
<b>Funktionsbeschreibung</b>	liest ein ByteWord vom Array
<b>Parameter</b>	Y = Arraypointer
<b>Rückgabewerte</b>	XL/X = Inhalt der Arrayzelle
<b>Register</b>	R21

<b>Funktionsname</b>	<b>api_awrite</b>
<b>Aufruf ohne Makro</b>	call api_awrite
<b>Funktionsbeschreibung</b>	schreibt ein Byte/Word ins Array
<b>Parameter</b>	Y = Arraypointer, XL/X = zu schreibender Wert
<b>Rückgabewerte</b>	—
<b>Register</b>	R21

<b>Funktionsname</b>	<b>api_pageset</b>
<b>Aufruf ohne Makro</b>	call api_pageset
<b>Funktionsbeschreibung</b>	setzt die Array-page für externen Speicher
<b>Parameter</b>	XL = neue Array-Page (nach Systemstart 0)
<b>Rückgabewerte</b>	—
<b>Register</b>	—