AVR-ChipBasic2: Interna

V1.39 (c) 2006-2011 Jörg Wolfram

1 Organisation des Videospeichers

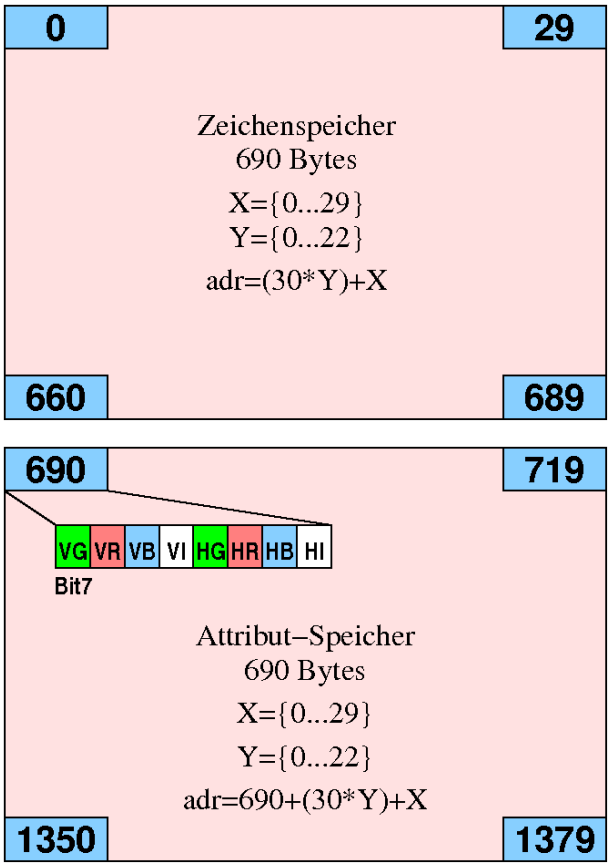
1.1 Unterschiede bei 8 und 16 Farben

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

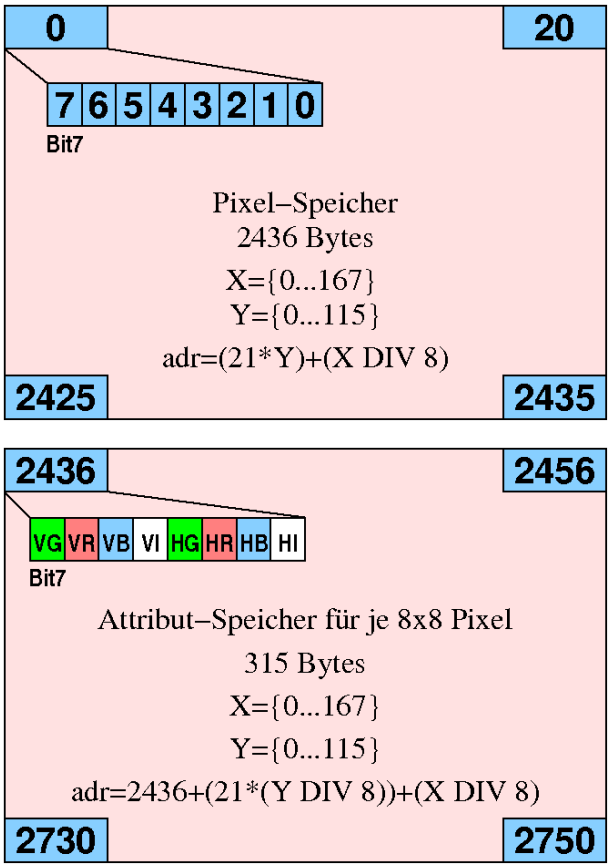


Für die Pseudografik sind die Zeichen 0...15 in 4 „Pixel” aufgeteilt. Bei 23 Zeilen a 30 Zeichen ergibt sich so eine Arbeitsfläche von 60x46 Punkten.

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

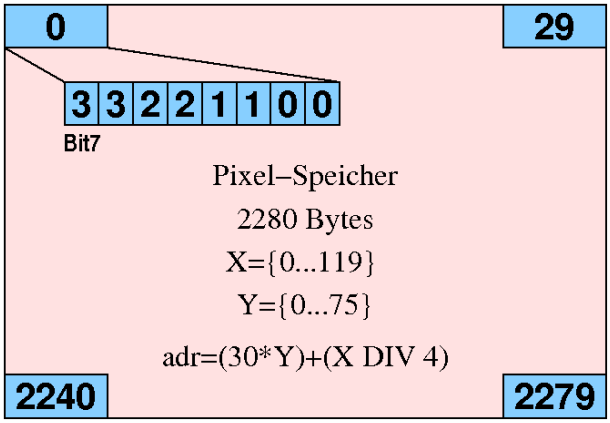


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entspricht Bit 0 dem Pixel ganz links.

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

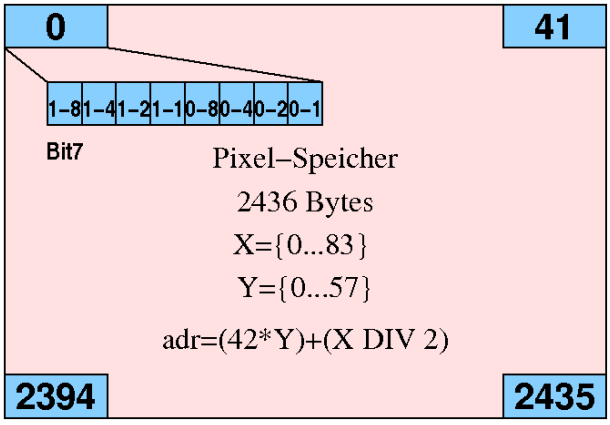


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entsprechen Bit 0 und 1 dem Pixel ganz links. Jedes Pixel wird durch zwei Bits repräsentiert, die auf einen der ersten 4 Paletteneinträge zeigen. Der Paletteneintrag bestimmt dann die Farbe.

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

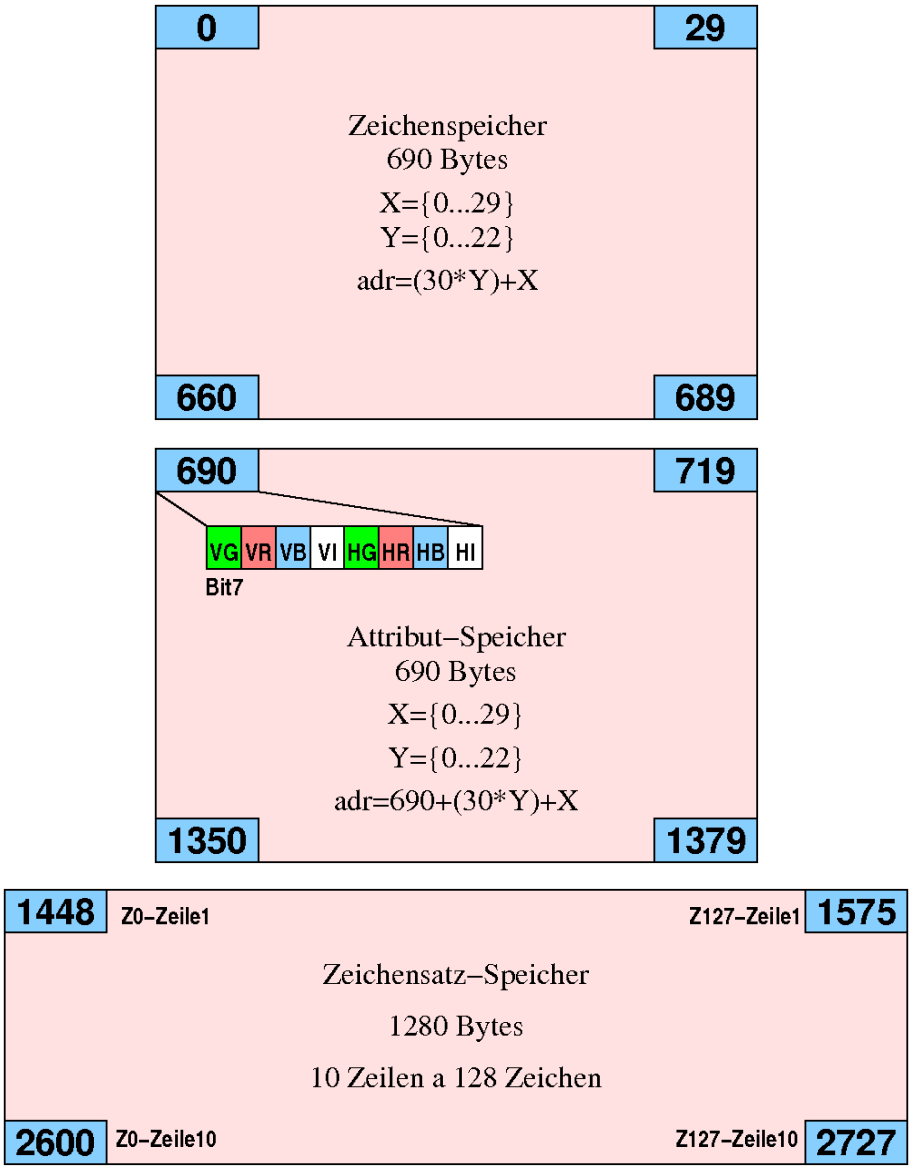


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Jedes Byte ist in 2 Nibbles zu 4 Bit unterteilt, die auf einen der 16 Paletteneinträge zeigen. Der Paletteneintrag bestimmt dann die Farbe.

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

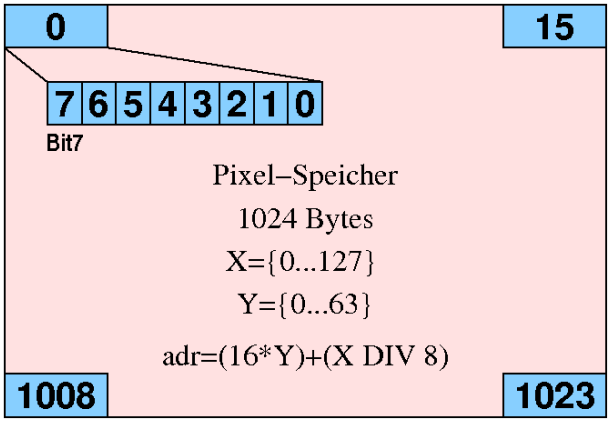


Im Zeichensatz sind die Bytes nach Zeichenzeilen geordnet, also 128 Bytes für die erste Zeichenzeile, dann 128 Bytes für die zweite Zeichenzeile...
Dabei sind die Pixel innerhalb einer Zeichenzeile noch differentiell gespeichert, Bit 7 ist das Pixel ganz links, ist Bit 6 ”0”, hat das nächste Pixeldie 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

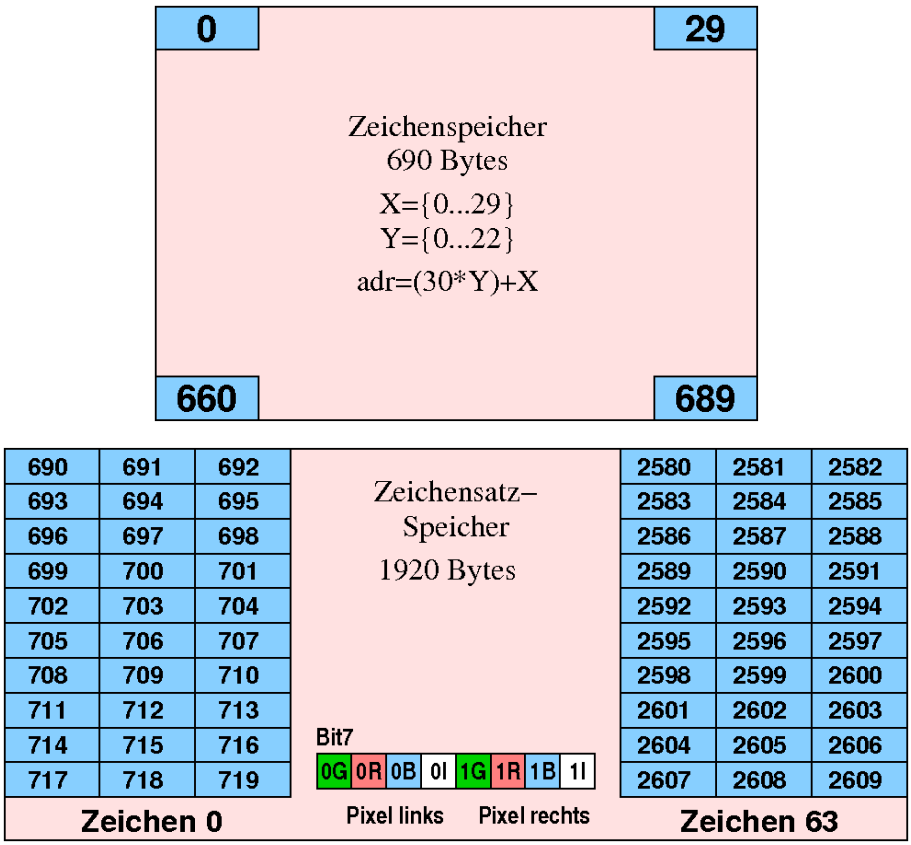


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entspricht Bit 0 dem Pixel ganz links. Für jedes Pixel gibt es ganau ein Bit, welches auf einen der ersten beiden Paletteneinträge zeigt. Der Paletteneintrag bestimmt dann die Farbe.

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 "Apllication 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_orom                           ;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 **Treiber**
x100	grün	Vorgabe für **Bibliotheken**
x101	cyan	Vorgabe für **Binärprogramme**
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 gegenn-über dem Tabellenanfang:

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

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	Bibliotheksnummer, 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	Bibliotheksnummer
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 habe 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 wiichtigsten 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

Funktionsname	Name der API-Funktion, entspricht auch dem Makro-Aufruf
Aufruf ohne Makro	Aufruf der Funktion, wenn ohne Makros gearbeitet wird
Funktionsbeschreibung	Kurzbeschreibung der Funktion
Parameter	Registerzuordnung zu den Parametern
Rückgabewerte	Registerzuordnung zu den Rückgabewerten
Register	Register, deren Inhalt verändert wird (ausser Rückgabewerte)
Videomodi	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

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

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

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

Funktionsname	**api_popxyz**
Aufruf ohne Makro	jmp api_popregs
Funktionsbeschreibung	holt die Register X,Y und Z sowie R20-R23 vom Stack und führt ein RET aus
Parameter	—
Rückgabewerte	—
Register	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.

Funktionsname	**api_add**
Aufruf ohne Makro	call api_add
Funktionsbeschreibung	X = X + R16/R17
Parameter	X, R16, R17
Rückgabewerte	X
Register	R18

Funktionsname	**api_sub**
Aufruf ohne Makro	call api_sub
Funktionsbeschreibung	X = X - R16/R17
Parameter	X, R16, R17
Rückgabewerte	X
Register	R16/R17 = -R16/R17

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

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

Funktionsname	**api_sin**
Aufruf ohne Makro	call api_sin
Funktionsbeschreibung	X = 256 * SIN(X)
Parameter	X (in Grad)
Rückgabewerte	X
Register	Z, R16, R17

Funktionsname	**api_cos**
Aufruf ohne Makro	call api_cos
Funktionsbeschreibung	X = 256 * COS(X)
Parameter	X (in Grad)
Rückgabewerte	X
Register	Z, R16, R17

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

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

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

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

Funktionsname	**api_abs**
Aufruf ohne Makro	call api_abs
Funktionsbeschreibung	gibt den Absolutwert von X zurück
Parameter	X
Rückgabewerte	X
Register	—

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

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

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

Funktionsname	**api_scale**
Aufruf ohne Makro	call api_scale
Funktionsbeschreibung	Skalierungsfunktion
Parameter	Die Parameter liegen in dem mit **api_getpartab** definierten Bereich
Rückgabewerte	R16/R17 = Resultat
Register	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:

Offset	**Parameter**
0	Y0
2	Y1
4	X0
6	X
8	X1

3.4 Programmsteuerung

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

Funktionsname	**api_fast**
Aufruf ohne Makro	call api_fast
Funktionsbeschreibung	schaltet die Bildausgabe ab
Parameter	—
Rückgabewerte	—
Register	—

Funktionsname	**api_slow**
Aufruf ohne Makro	call api_slow
Funktionsbeschreibung	schaltet die Bildausgabe ein
Parameter	—
Rückgabewerte	—
Register	—

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

3.5 Auskunftsfunktionen

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

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

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

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

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

Funktionsname	**api_getvar**
Aufruf ohne Makro	call api_getvar
Funktionsbeschreibung	bestimmt die Anfangsadresse der BASIC-Variablen
Parameter	—
Rückgabewerte	Y zeigt auf den Anfang der BASIC-Variablen (Variable ”A”)
Register	—

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

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

Funktionsname	**api_getvalues**
Aufruf ohne Makro	call api_getvalues
Funktionsbeschreibung	bestimmt die Anfangsadresse des Funktionsparameter-Blocks
Parameter	—
Rückgabewerte	Y zeigt auf den Anfang des Blockes
Register	—

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

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

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

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

- 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.

Funktionsname	**api_getprog**
Aufruf ohne Makro	call api_getprog
Funktionsbeschreibung	bestimmd die Nummer des aufrufenden Programmes im Flash
Parameter	—
Rückgabewerte	ZL = Programmnummer (0. . . 7)
Register	ZH

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

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

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

3.6 Tastatur

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

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

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

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

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

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

3.7 Bildschirmausgabe

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

Funktionsname	**api_clearvec**
Aufruf ohne Makro	call api_clearvec
Funktionsbeschreibung	löscht den Vektorbereich im Grafikmode 7
Parameter	—
Rückgabewerte	—
Register	—
Videomodi	7

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

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

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

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

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

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

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

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

Funktionsname	**api_thistext**
Aufruf ohne Makro	call api_thistext
Funktionsbeschreibung	gibt einen nullterminierten String aus dem Flash aus, der direkt auf den Aufruf folgt.
Parameter	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.
Rückgabewerte	—
Register	Z, R20, R21
Videomodi	alle

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

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

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

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

Funktionsname	**api_copychar4**
Aufruf ohne Makro	call api_copychar4
Funktionsbeschreibung	kopiert ein Zeichen aus dem System-Zeichensatz in den User-Zeichensatz für Videomode 4
Parameter	R20 = System-Zeichen, R21 = User Zeichen (0. . . 127)
Rückgabewerte	—
Register	—
Videomodi	4

Funktionsname	**api_copychar6**
Aufruf ohne Makro	call api_copychar6
Funktionsbeschreibung	kopiert ein Zeichen aus dem System-Zeichensatz in den User-Zeichensatz für Videomode 6
Parameter	R20 = System-Zeichen, R21 = User Zeichen (0. . . 63), R22 = VG/HG Farbe
Rückgabewerte	—
Register	—
Videomodi	6

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

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

3.8 Grafik

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

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

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

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

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

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

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

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

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

3.9 Kommunikation

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

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

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

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

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

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

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

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

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

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

3.10 Audio

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

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

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

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

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

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.

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

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

3.12 Funktionen für den I2C-Bus

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

Funktionsname	**api_i2c_start**
Aufruf ohne Makro	call api_i2c_start
Funktionsbeschreibung	erzeugt eine Start-Kondidion auf dem I2C-Bus
Parameter	—
Rückgabewerte	—
Register	R20

Funktionsname	**api_i2c_stop**
Aufruf ohne Makro	call api_i2c_stop
Funktionsbeschreibung	erzeugt eine Stop-Kondidion auf dem I2C-Bus
Parameter	—
Rückgabewerte	—
Register	R20

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

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

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

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

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

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

3.13 Alert- und Abfrageboxen

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

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

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

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

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

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

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

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

3.14 Zugriff auf das Dataflash-Dateisystem

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

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

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

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

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

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

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

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

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

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

Funktionsname	**api_fs_check**
Aufruf ohne Makro	call api_fs_check
Funktionsbeschreibung	testet das Dataflash auf gültiges Dateisystem
Parameter	
Rückgabewerte	
Register	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.

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

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

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

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

3.15 Direktzugriff auf das Dataflash

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

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

3.16 BASIC/SYSTEM-Funktionen

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

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

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

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

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

Funktionsname	**api_exparser**
Aufruf ohne Makro	call api_exparser
Funktionsbeschreibung	interpretiert eine Formel im Textformat
Parameter	Z = Zeiger auf Funktionstext
Rückgabewerte	X = Resultat (16 Bit signed)
Register	Y, Z, R4…R7, R20…R23

Funktionsname	**api_tparser**
Aufruf ohne Makro	call api_tparser
Funktionsbeschreibung	stellt einen Tokenparser zur Verfügung
Parameter	Z = Zeiger auf Tokentabelle im Flash, Y = Zeiger auf Text im RAM, R22 = Länge eines Eintrages in Bytes
Rückgabewerte	R20=gefundener Token oder 0
Register	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 eine Eintrages 6 Bytes (1 Byte Tokenwert und 5 Bytes Text):

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

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

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.

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

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

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

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

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

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

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.

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

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

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