

# AVR-ChipBasic2: Interna

## V1.03 (c) 2006-2009 Jörg Wolfram

### 1 Organisation des Videospeichers

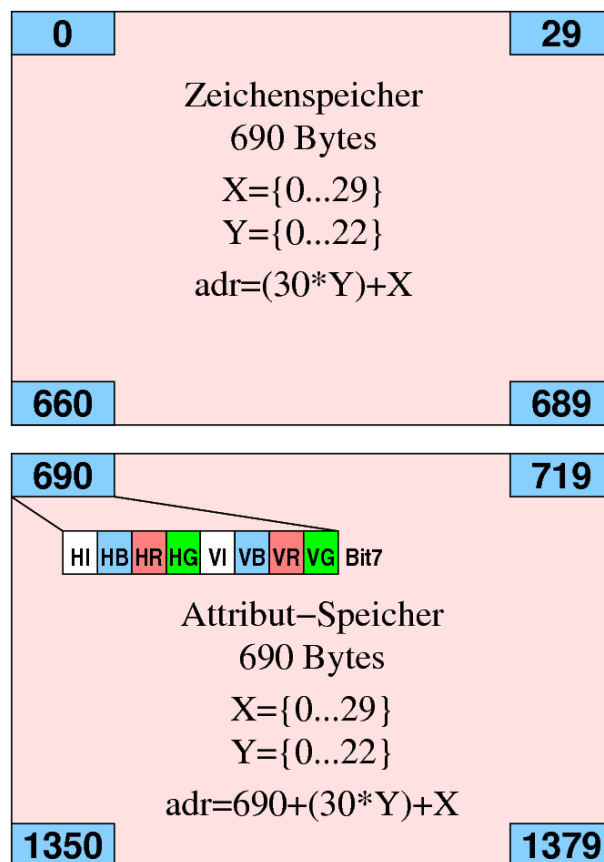
#### 1.1 Unterschiede bei 8 und 16 Farben

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

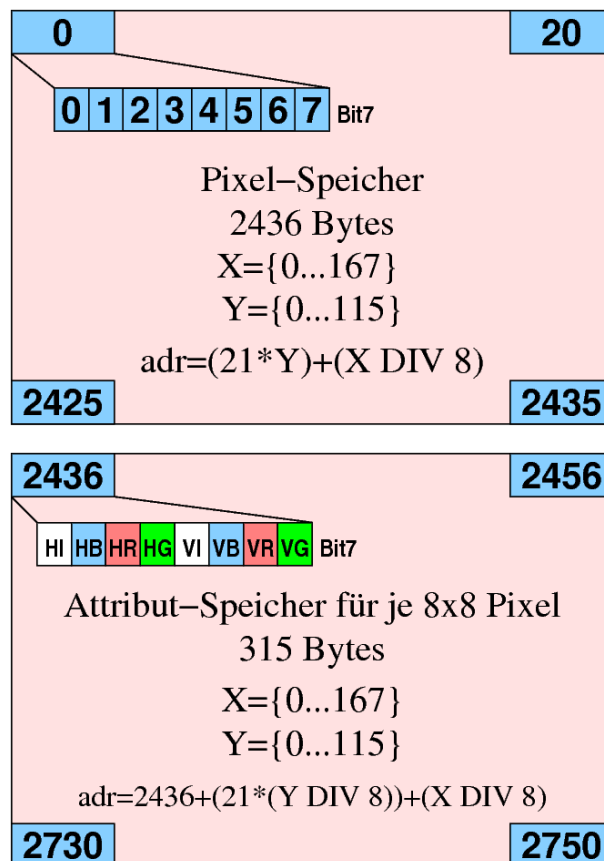


Für die Pseudografik sind die Zeichen 0...15 in 4 „Pixel“ aufgeteilt. Bei 23 Zeilen a 30 Zeichen ergibt sich so eine Arbeitsfläche von 60x46 Punkten.

### 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

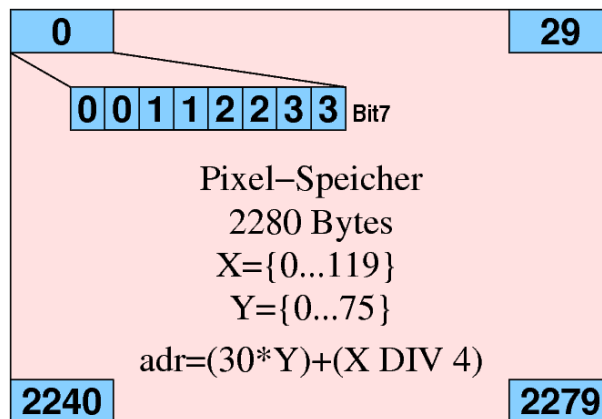


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entspricht Bit 0 dem Pixel ganz links.

## 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

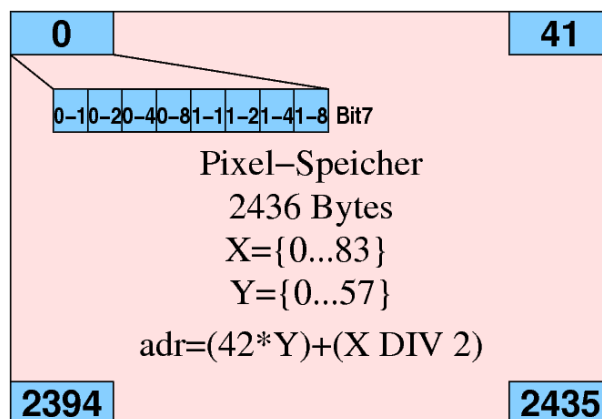


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entsprechen Bit 0 und 1 dem Pixel ganz links. Jedes Pixel wird durch zwei Bits repräsentiert, die auf einen der ersten 4 Paletteneinträge zeigen. Der Paletteneintrag bestimmt dann die Farbe.

## 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

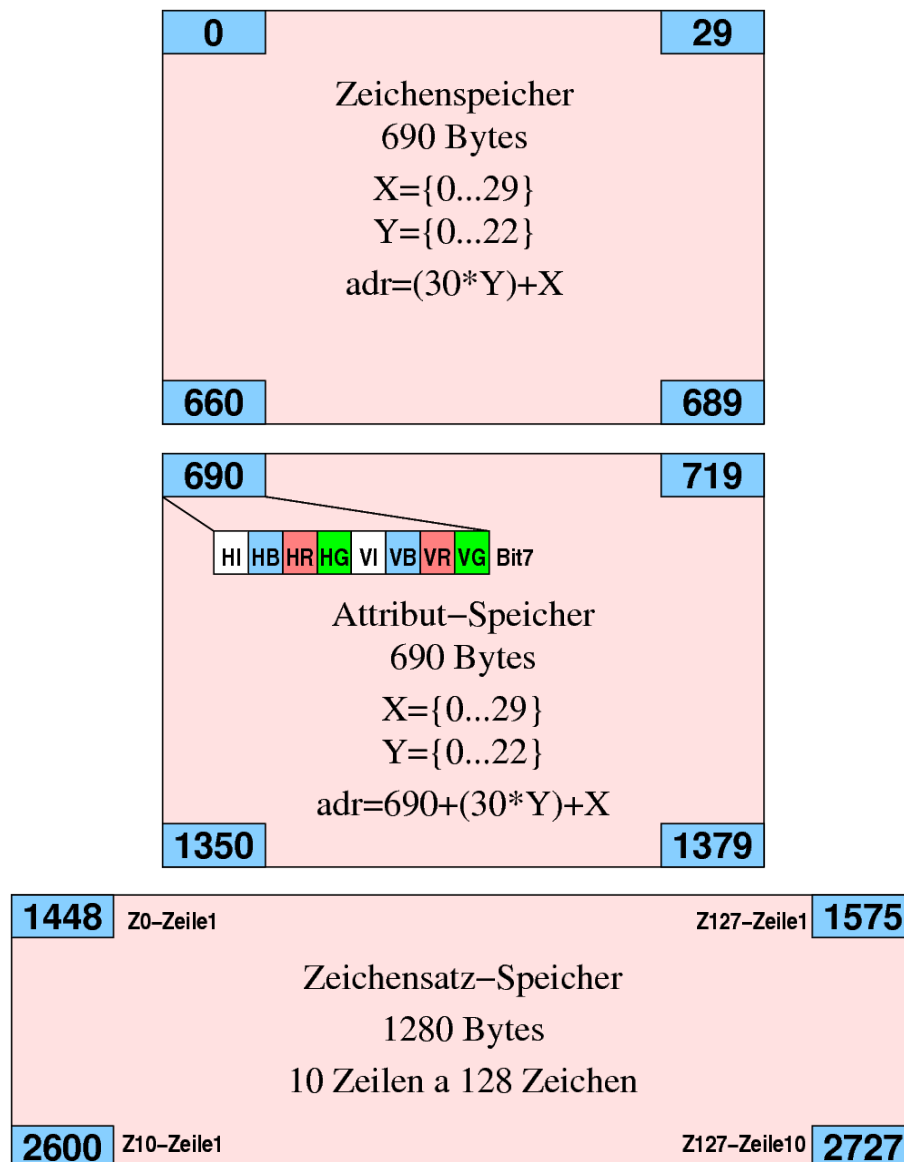


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Jedes Byte ist in 2 Nibbles zu 4 Bit unterteilt, die auf einen der 16 Paletteneinträge zeigen. Der Paletteneintrag bestimmt dann die Farbe.

## 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



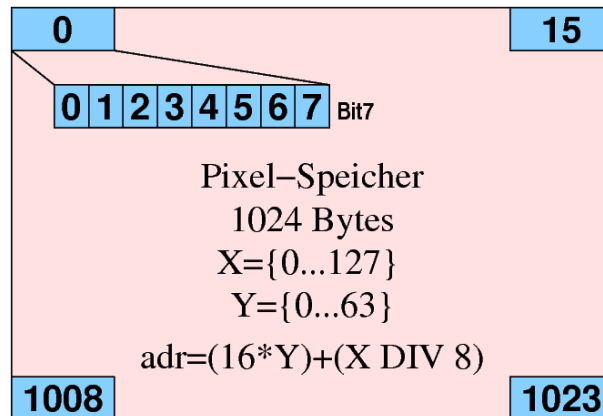
Im Zeichensatz sind die Bytes nach Zeichenzeilen geordnet, also 128 Bytes für die erste Zeichenzeile, dann 128 Bytes für die zweite Zeichenzeile...

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

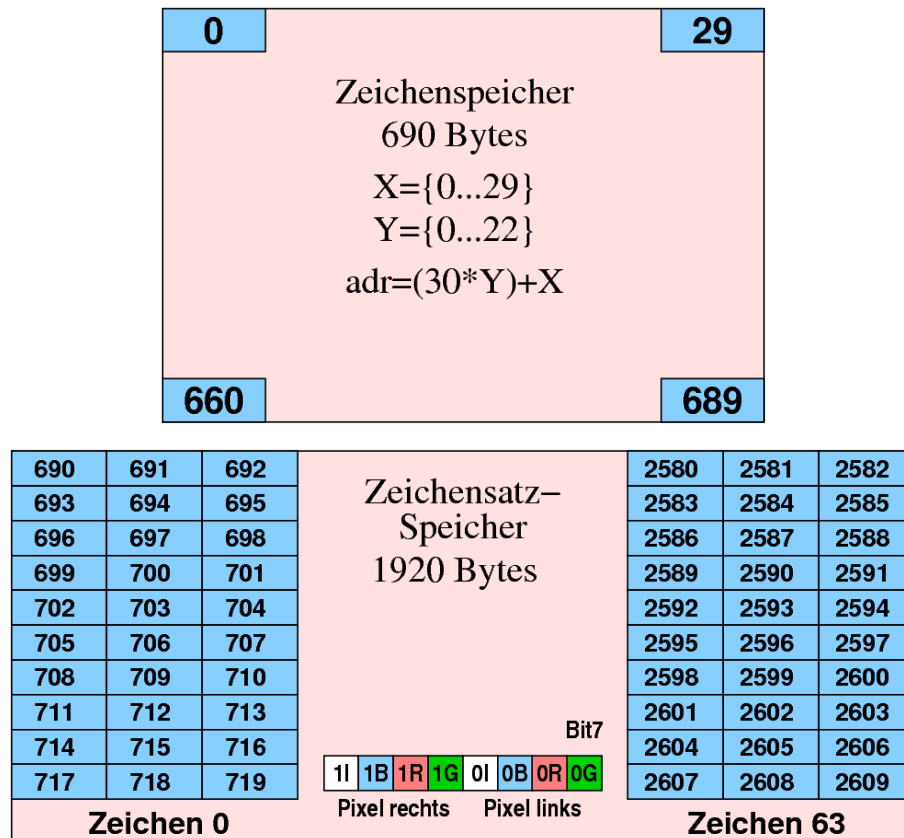


Die Pixelbytes sind von links nach rechts sowie von oben nach unten organisiert. Innerhalb des Bytes entspricht Bit 0 dem Pixel ganz links. Für jedes Pixel gibt es genau ein Bit, welches auf einen der ersten beiden Paletteneinträge zeigt. Der Paletteneintrag bestimmt dann die Farbe.

## 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



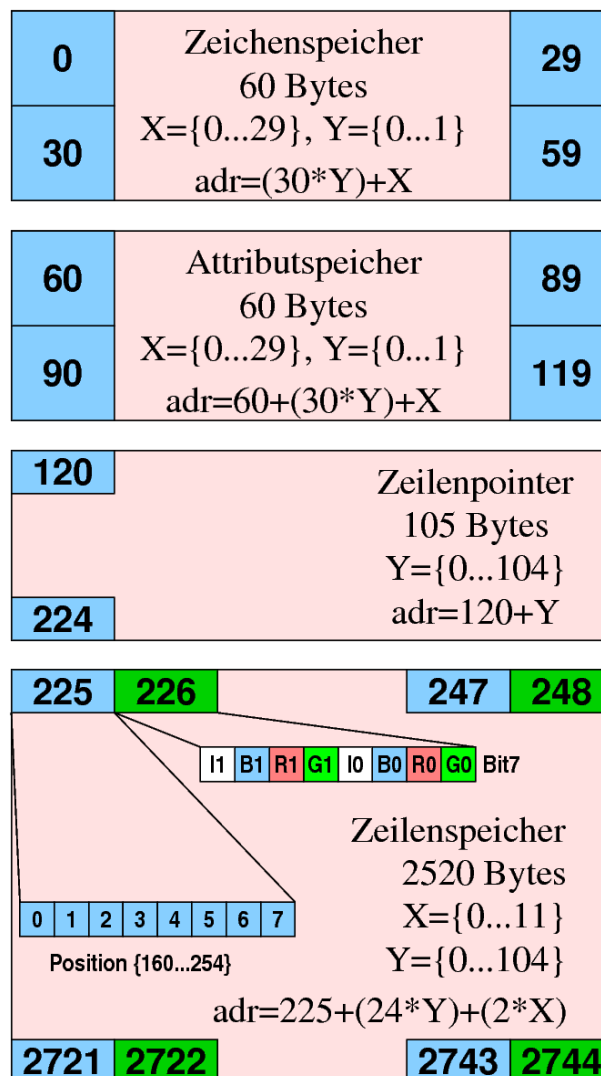
Im Zeichensatz liegen die 30 Bytes jedes Zeichens direkt hintereinander. Jedes Pixel wird durch drei Bits repräsentiert, innerhalb des Bytes entsprechen Bit 1...3 dem rechten und Bit 5...7 dem linken Pixel. Drei Bytes ergeben eine Zeichenzeile von 6 Pixeln, ein Zeichen besteht aus 10 dieser Zeichenzeilen.

## 1.9 Videomode 7

Diesen Modus könnte man als eine Art "Vektormodus" bezeichnen. Am oberen und unteren Bildschirmrand gibt es jeweils eine Textzeile, die sich wie die ersten beiden Zeilen im Modus 0 verhalten. Dann hört aber die Ähnlichkeit schon auf. Danach kommen 105 Byte-Zeiger, gefolgt von 105 Zeilendefinitionen à 24 Bytes. Jeder der Zeiger gibt an, welche der 105 Zeilendefinitionen für die gerade aktuelle Zeile relevant ist. Somit können alle Zeilen ihre eigene Definition oder auch alle Zeilen die gleiche Definition haben. Werte größer als 104 sollten nicht eingestellt werden, da dann das Verhalten nicht definiert ist.

Der Bildspeicher ist folgendermaßen aufgeteilt:

Anfangsadresse	Endadresse	Bytes	Funktion
0	59	60	2 Zeilen Zeichen
60	119	60	2 Zeilen Attribute
120	224	105	105 Zeilenpointer
225	2744	2520	105 Zeilendefinitionen
2745	2759	15	ungenutzt



Die Zeilendefinitionen bestehen aus 12 Einträgen zu jeweils 2 Bytes. Das erste Byte bestimmt den Abstand zum linken Bildschirmrand wozu noch 160 addiert werden muß. Höchster Wert ist 254, so dass die horizontale Auflösung 95 Pixel beträgt. Das zweite Byte gibt die anzuzeigenden Farben an. Dabei werden im Wechsel Die Farbwerte aus dem oberen und dem unteren Nibble angezeigt. Zu beachten ist noch, dass der kleinste Abstand 160 sein muß und jeder folgende Abstandswert mindestens um 1 größer als der vorherige.

## 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 (mit dem 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 3 weitere Bytes 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 eingebundene Tabellen zuzugreifen. Allerdings ist das über das API möglich:

```
.org          0x4000
start:        .db "TestprogrammN",0xff,0xff,0xff

lese:         api_getvram                ;setzt Y auf den Anfang des Bildspeichers
              api_getbase                ;setzt Z auf Anfang des Programmes
              ldi XL,LOW(daten-start)    ;Bestimmung des Offsets
              ldi XH,HIGH(daten-start)
              call api_dataptr            ;setzt Z auf die Adresse von "daten:"
lese_1:        lpm r16,Z+
              cpi r16,0
loop:         breq loop
              st Y+,r16
              rjmp lese_1

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

Dieses Programm schreibt "Mein Test" in die linke obere Bildschirmecke, allerdings geht so etwas auch viel einfacher:

```
.org          0x4000
start:        .db "TestprogrammN",0xff,0xff,0xff

lese:         api_thistext
daten:        .db 0,0,"Mein Text",0
loop:         rjmp loop
```

Am Anfang müssen 12 bytes Programmname und ein "N" (0x4E) stehen, danach folgen 3 reservierte Bytes (0xFF). Das eigentliche Programm beginnt mit einem Offset von 16 Bytes (8 Words).

Es ist nicht möglich, eigene Interrupt-Routinen zu einzubinden und ein Sperren der Interrupts hat die Abschaltung der Bild- und Tonausgabe zur Folge. Weiterhin sollten die Register r8 (aktuelle Bildschirmzeile LOW) und r9 (aktuelle Bildschirmzeile HIGH) nach Möglichkeit nicht beschrieben werden.



## 2.3 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 unter anderen Betriebssystemen nur eingeschränkt oder nicht funktionieren. Folgende Schritte sind dabei 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. Erzeugung einer beliebigen Datei, welche zusammen mit der Binärdatei aus dem vorigen Schritt genau 3072 Bytes lang ist
6. Die erzeugte 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 im **API**-Verzeichnis ein kleines Build-Script, welches einen Großteil der Arbeit abnimmt. Aufgerufen wird es mit **./build.pl 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 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.

Zusätz-

lich 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. R24 ist nach der Operation entweder 0 (kein Fehler) oder enthält einen Fehlercode. Die Fehlercodes entsprechen denen im BASIC Programm. Dabei bildet das X-Register eine Art "Akkumulator", als zweiter Operand wird das Registerpaar R16/R17 genutzt.

<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>	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>	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>	—

### 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>	—

### 3.5 Auskunftsfunktionen

<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_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_getbuffer1
<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 à 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_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>	—



### 3.6 Einstellungen

<b>Funktionsname</b>	<b>api_setcolor</b>
<b>Aufruf ohne Makro</b>	call api_setcolor
<b>Funktionsbeschreibung</b>	stellt Vorder- und Hintergrundfarbe ein
<b>Parameter</b>	XL (LOW-Nibble=FG, HIGH-Nibble=bg [igrbIGRB])
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_setfcolor</b>
<b>Aufruf ohne Makro</b>	call api_setfcolor
<b>Funktionsbeschreibung</b>	stellt Vordergrundfarbe ein, Hintergrundfarbe bleibt erhalten
<b>Parameter</b>	XL (LOW-Nibble=FG [xxxxIGRB])
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_setborder</b>
<b>Aufruf ohne Makro</b>	call api_setborder
<b>Funktionsbeschreibung</b>	stellt die Randfarbe ein
<b>Parameter</b>	XL (LOW-Nibble=Farbe [xxxxIGRB])
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_setvmode</b>
<b>Aufruf ohne Makro</b>	call api_setvmode
<b>Funktionsbeschreibung</b>	stellt den Videomode ein
<b>Parameter</b>	XL
<b>Rückgabewerte</b>	—
<b>Register</b>	X

<b>Funktionsname</b>	<b>api_setpalette</b>
<b>Aufruf ohne Makro</b>	call api_setpalette
<b>Funktionsbeschreibung</b>	setzt einen Paletteneintrag
<b>Parameter</b>	XL (LOW-Nibble=Farbe [xxxxIGRB]), XH Palettenindex (0...15)
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_setchannel</b>
<b>Aufruf ohne Makro</b>	call api_setchannel
<b>Funktionsbeschreibung</b>	ändert den Ausgabekanal (siehe PRINT in der BASIC-Referenz)
<b>Parameter</b>	XL = Ausgabekanal
<b>Rückgabewerte</b>	—
<b>Register</b>	—

<b>Funktionsname</b>	<b>api_setfont</b>
<b>Aufruf ohne Makro</b>	call api_setfont
<b>Funktionsbeschreibung</b>	ändert den Textfont in den Grafikmodi
<b>Parameter</b>	XL = Font (0/1, es wird nur Bit 0 ausgewertet)
<b>Rückgabewerte</b>	—
<b>Register</b>	—

### 3.7 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>	—

### 3.8 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_copychar4</b>
<b>Aufruf ohne Makro</b>	call api_copychar4
<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_copychar6</b>
<b>Aufruf ohne Makro</b>	call api_copychar6
<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

### 3.9 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

### 3.10 Kommunikation

<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_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_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_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



### 3.11 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>	—

### 3.12 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

### 3.13 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

### 3.14 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>	—

### 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-Funktionen

<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