# Ansteuerung des VGA-Anschlusses über Atmega

Tobias Zimmer

12. August 2007

# Inhaltsverzeichnis

# 1 Einleitung

Dieser Vortrag befasst sich mit der Ansteuerung eines Bildschirms über einen VGA-Anschluss mit einem Atmel- Atmega Prozessor. Hier liegt vorallem die Machbarkeit im Vordergrund. Dazu habe ich mir auch die Fragen gestellt, welche Version des Atmega am sinnvollsten für diese Realisierung sein könnte, da die Bildwiedergabe ein zeit- und vorallem speicherintensiver Prozess ist. Zu Anfang folgen ersteinmal ein paar grundlegende Informationen zum Video Graphics Array, danach beschäftige ich mich mit den verschiedenen Realisierungsmöglichkeiten.

## 1.1 VGA-Standard

Der VGA-Standard wurde 1987 von IBM eingeführt. Er ist der Nachfolger der beiden Grafikstandards EGA (Enhanced Graphics Adapter) und CGA (Color Graphics Adapter), die beide ebenfalls von IBM stammen. VGA steht für Video Graphics Array. Aufgrund der damaligen Monopolstellung von IBM hält sich VGA immernoch als minimal verfügbarer Standard, den jede heutige Grafikkarte unterstützt. Windows nutzt in aktuellen Versionen heute immernoch den VGA-Standard als „treiberunabhängige“ Voreinstellung, wärend verschiedene Linux-Versionen bereits ohne zusätzliche Treiber sämtliche VESA-Modi unterstützen. Meist wird der Begriff VGA auch nur für die Auflösung 640x480 genutzt, obwohl eine VGA-Grafikkarte natürlich mehr als diese eine Auflösung kann.

## 1.2 VGA-Grafikkarte

Die wichtigsten Merkmale einer VGA-Grafikkarte sind zum einen die mindestens 256 kByte Video-RAM, zum anderen die Farbunterstützung bis 256 Farben. Bei 60 Vollbildern in der Sekunde ist es möglich, 480 Zeilen pro Bild darzustellen, bei 70 sind es immerhin noch 400 Zeilen. Pro Zeile sind abhängig vom Pixeltakt bis zu 720 Pixel möglich. Meist jedoch wird hier bei einem Takt von 25,179 MHz eine Breite des Bildes von 640 Pixeln verwendet. Die Horizontalfrequenz liegt konstant bei 31,5 KHz. Jede VGA-Grafikkarte besitzt eine Farb- und eine Texttabelle, die im folgenden näher beschrieben werden. Wichtig ist, dass eine solche Karte nicht zwingend eine VGA-Buchse benötigt, um VGA-kompatibel zu sein. Genausowenig muss ein Gerät mit einer VGA-Buchse eine VGA-Unterstützung aufweisen.

### 1.2.1 Farbtabelle

Die Farbtabellen bei VGA-Grafikkarten besitzen immer die folgenden Eigenschaften:

- Für den Index der Farbtabelle wird ein Byte reserviert. Daraus folgt, dass $2^8 = 256$ Indizes für die Tabelle zur Verfügung stehen.

- Die Farben pro Eintrag sind unterteilt in Werte für Rot, Grün und Blau.
- Für diese 3 Farben stehen jeweils 6 Bits zur Verfügung, dh. pro Eintrag benötigt man 18 Bit.
- Insgesamt ergeben sich dadurch $2^18 = 262144$ Möglichkeiten an verschiedenen Farben. Diese lassen sich beliebig in die 256 Einträge der Tabelle einfügen.
- Aufgrund geforderter Abwärtskompatibilität zu EGA hat man sich dafür entschieden, die Farbtabelle in 4 kleinere Tabellen mit 64 Einträgen zu splitten. Durch schnelles Umschalten zwischen diesen Komponenten bleibt aber dennoch eine komplette Nutzungsmöglichkeit der 256 Farben erhalten.

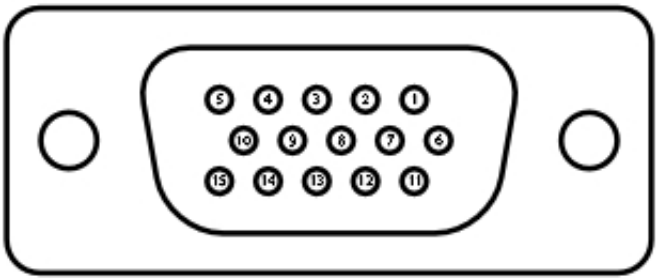
### 1.2.2 Textmodus

Durch den Textmodus kann man den Speicherverbrauch einer VGA-Grafikkarte noch weiter drücken:

- Man benötigt eine Zeichentabelle, die wie die Farbtabelle fest im System verankert ist.
- Ein Beispiel für die Nutzung des Textmodus in Betriebssystemen bietet hier DOS, wobei man zB. in verschiedenen Dosshells sogar versucht hat, eine grafische Benutzeroberfläche mithilfe einer Zeichentabelle zu erstellen.
- Die Texttabelle enthält Zeichen der Auflösung 8x8 bis 8x16 Pixel.
- Daraus ergeben sich bei 720x400 Pixeln Bildgröße Auflösungen von 80x25 bis 80x50 Zeichen pro Bild.
- Der Vorteil dieses Modus ist wie oben bereits erwähnt die wesentlich geringere Nutzung des Grafikspeichers auf Kosten der Darstellungsqualität.

## 1.3 VGA-Anschluss

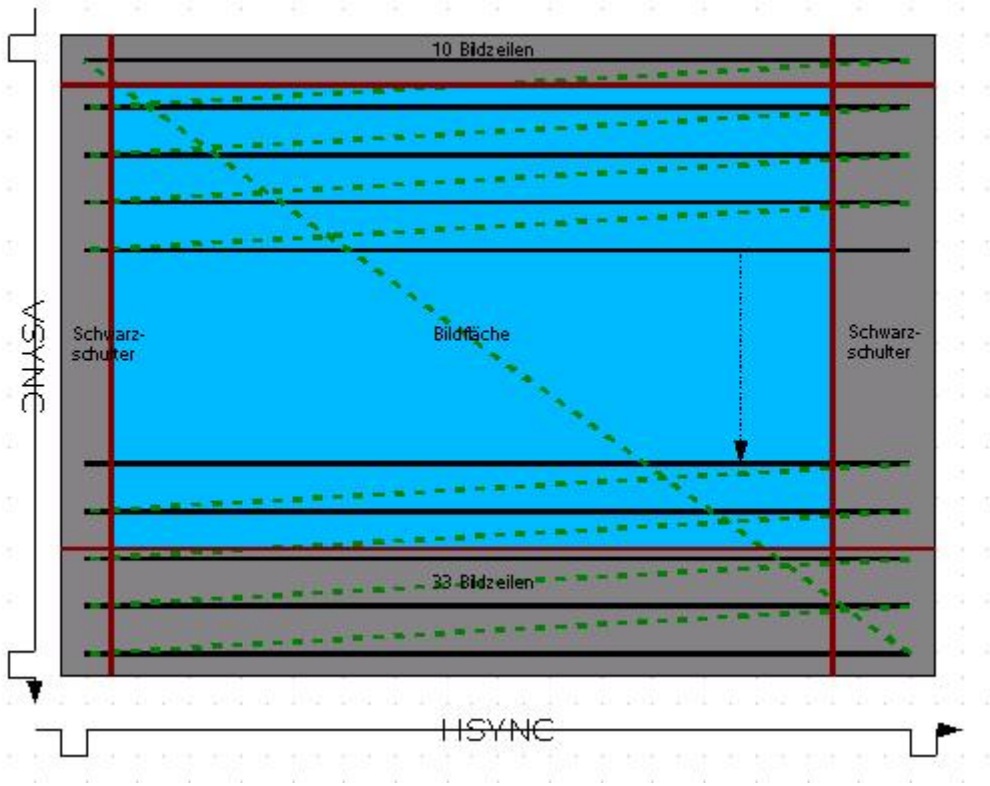
Der VGA-Anschluss besitzt 15 Pins, von denen der 9te jedoch meistens nicht belegt oder garnicht erst vorhanden ist. Die ersten 3 Pins belegen die Kanäle für Rot, Grün und Blau. Die zugehörigen Masseleitungen finden sich auf den Pins 6 bis 8. Die erforderlichen Synchronisationssignale finden sich bei Pin 13 und 14, die entsprechende Masse auf 10. Pin 4, 11 und 12 führen Identifikations-Bits. Über Pin 12 können auch digitale Daten übertragen werden. Der digitale Takt sitzt auf Pin 15, die digitale Masse auf Pin 5. Der Vortrag beschränkt sich aber auf die Pins zur Übertragung der Farben und auf die Synchronisation der Bildwiedergabe.




| PIN | Funktion |
|---|---|
| 1 | Rot |
| 2 | Grün |
| 3 | Blau |
| 4 | - / ID2 |
| 5 | GND (digitale Masse) |
| 6 | Rot Masse |
| 7 | Grün Masse |
| 8 | Blau Masse |
| 9 | - / kein PIN / +5V |
| 10 | GND (Masse SYNC) |
| 11 | ID0 |
| 12 | ID1 / DCC - SDA |
| 13 | HSYNC |
| 14 | VSYNC |
| 15 | Takt / DCC - SCL |

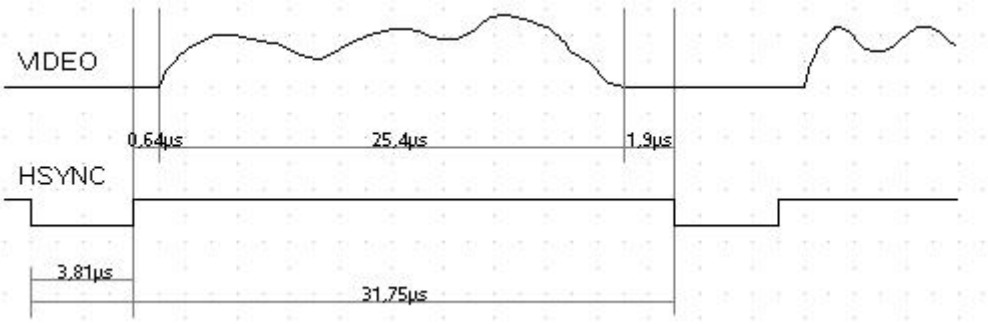
### 1.4 Synchronisation

Synchronisiert wird der Bildschirm mithilfe von zwei Leitungen, der horizontalen Synchronisation (HSYNC) und der vertikalen Synchronisation (VSYNC). Die Synchronisation findet bei VGA jeweils immer bei fallender Flanke statt, was auch bedeutet, dass auf den beiden Leitungen im Normalfall High-Pegel anliegt. VSYNC ist für die Synchronisation des kompletten Bildes, HSYNC für die einer Zeile zuständig.




### 1.4.1 horizontale Synchronisation

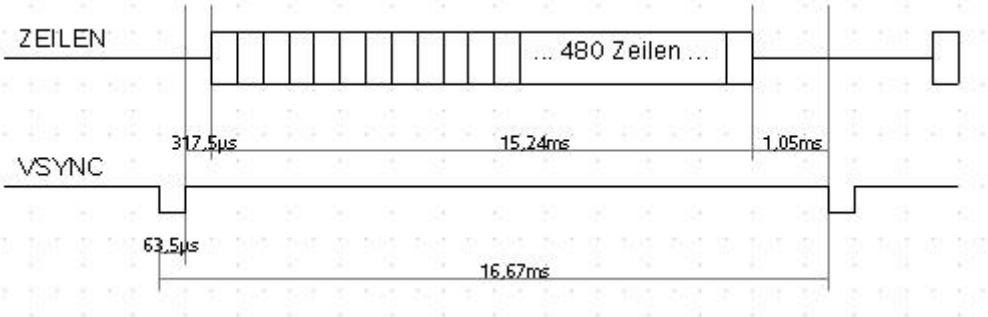
Die horizontale Synchronisation sorgt dafür, dass der Bildschirm erfährt, wann ein Sprung in die nächste Zeile notwendig wird. Da dieser Sprung Zeit kostet, muss das Signal eine größere zeitliche Ausdehnung besitzen. Außerdem beginnt die Zeile ein Stück vor dem eigentlich dargestellten Bild. Dieser nicht dargestellte Bereich wird vordere Schwarzschulter genannt. Am Ende der Zeile befindet sich ebenfalls wieder ein solcher Bereich, der analog dazu hintere Schwarzschulter heißt. Die Gesamtlänge des Signals beträgt bei einem Pixeltakt von 25,179 und 800 Pixeln pro Zeile MHz 31,75 $\mu$s. Von diesen 800 Pixeln Gesamtlänge muss man entsprechend 96 für das Synchronisationssignal, 16 für die vordere und 48 für die hintere Schwarzschulter abziehen, wodurch man dann auf eine Breite des sichtbaren Bilds von 640 Pixeln kommt.




- 31,75 $\mu$s Gesamtlänge: 800 Pixel
- 3,81 $\mu$s Synchronisationssignal: 96 Pixel
- 0,64 $\mu$s vordere Schwarzschulter: 16 Pixel
- 25,4 $\mu$s sichtbare Bildpunkte einer Zeile: 640 Pixel
- 1,9 $\mu$s hintere Schwarzschulter: 48 Pixel

### 1.4.2 vertikale Synchronisation

Die vertikale Synchronisation signalisiert dem Bildschirm, dass ein neues Bild beginnt. Auch hier benötigt der Bildschirm eine gewisse Zeit, damit der Elektronenstrahl des Bildschirms von der unteren rechten Ecke in die obere linke Ecke wandern kann. Von dort beginnt dann das nächste Bild. Bei 525 Bildzeilen sind das im gesamten 16,64 ms pro Bild bei 60 Bildern in der Sekunde. Auch hier entfallen 2 Zeilen auf das Synchronisationssignal, 10 Zeilen auf eine obere und 33 Zeilen auf eine untere Schwarzschulter. Die 480 übrigen Zeilen werden als sichtbares Bild auf dem Monitor dargestellt.

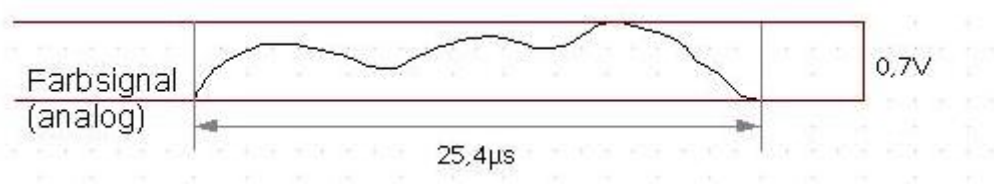



- 16,64 ms Bildhöhe: 525 Zeilen
- 63,5 $\mu$s Synchronisationssignal: 2 Zeilen

- 315,7 $\mu$s nicht sichtbar: 10 Zeilen
- 15,24 ms sichtbares Bild: 480 Zeilen
- 1,05 ms nicht sichtbar: 33 Zeilen

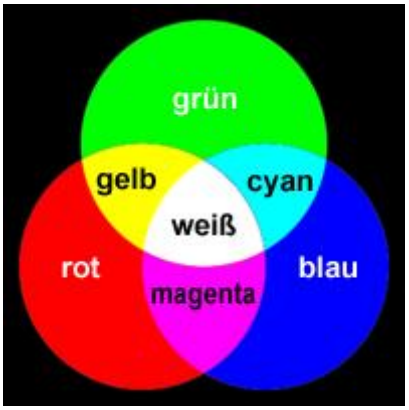
### 1.4.3 Farbdarstellung

In den 25,4 $\mu$s der sichtbaren Bildzeile werden an den drei Farbkanälen analoge Signale angelegt. Die Spannung, die wärend der Schwarzschulter anliegt, bestimmt den Pegel für Schwarz bzw. Dunkel. Bewegt sich die Spannung um 0,7 Volt nach oben, erhält man den hellstmöglichen Ton dieser Farbe. Innerhalb dieses Bereichs lässt sich die Farbe entsprechend variieren.




Die drei Farbkanäle für Rot, Grün und Blau sind unidirektional, dh. man bekommt keine Rückmeldung für eventuelle Fehler. Prinzipiell ist das Auge des Betrachters für die Fehlerbestimmung zuständig, da man bei einem Signal entweder ein korrektes Bild zu sehen bekommt oder ein fehlerhaftes. Der Innenwiederstand eines handelsüblichen Monitors beträgt 75 $\Omega$. Dieser Wert wird später herangezogen für die Berechnung der Vorwiederstände bei den einzelnen Konzepten.

Wie bei Monitoren üblich, wird die Farbe am Bildschirm durch additive Farbmischung erzeugt, bei Röhrenbildschirmen durch fluoreszierende Leuchtstoffstreifen, bei Flüssigkristallbildschirmen durch verschiedenfarbige Subpixel.



# 2 Lösungsansätze

Zu möglichen Lösungen eines VGA-Anschlusses unter Nutzung eines ATmega-Microprozessors muss man sich zunächst folgende Fragen stellen:

1. Reicht der Speicherplatz ?
2. Reicht die Zeit ?

## 2.1 Speicherbedarf

Die Frage nach dem Speicherplatz stellt sich dadurch, dass die Erzeugung eines Bilds auf einem Monitor erfahrungsgemäß viel Videospeicher benötigt, in dem das Bild vor der Übertragung an den Monitor gespeichert sein muss. Hier möchte ich zunächst von der üblichen 640x480-Pixel-Darstellung ausgehen.

### 2.1.1 Pixeldarstellung

- Eine Auflösung von 640x480 bedeutet umgerechnet eine Pixelanzahl von 307200
- Geht man zunächst von einer Schwarz-Weiß-Darstellung aus, bei der man für jedes Pixel ein Bit, nämlich für hell und dunkel, benötigt, kommt man auf 38400 Byte für ein einzelnes Bild
- Erhöht man die Farben auf $8 = 2^3$, dh. dehnt man die Schwarz-Weiß-Lösung auf die 3 Farbkanäle aus, erhält man 115200 Byte

Schon bei der Schwarz-Weiß-Darstellung wird der RAM eines ATmega- Prozessors nicht reichen. Der ATmega644 besitzt zB. nur 4kByte RAM, welche schon wesentlich kleiner sind als die 38,4 benötigten. Hier muss man also feststellen, dass eine Grafikkarte, die eine Bildabfolge mit der Auflösung 640x480 erzeugen kann und auf einem ATmega basiert, ohne zusätzlichen Arbeitsspeicher nicht realisierbar ist.

### 2.1.2 Textmodus

Abhilfe schafft hier ein Textmodus, wie ihn standardmäßig auch normale VGA-Grafikkarten unterstützen. Eine Zeichentabelle für 128 verschiedene Zeichen, die im Flashspeicher vorhanden ist, benötigt bei einer Auflösung von 8x16 Pixeln pro Zeichen genau 2048 Byte. Bei einem ATmega 16 mit 16kByte Flashspeicher wäre diese Tabelle also absolut machbar. Legt man wieder die Auflösung von 640x480 zugrunde, kommt man auf eine Anzahl von 28x30 Zeichen.

Die Zeilenanzahl von 30 Zeichen ist leicht zu ermitteln:

- 480 Zeilen entsprechen 480 Pixeln in der Höhe
- Ein Zeichen besitzt 16 Pixel Höhe
- $\frac{480}{16} = 30$ Zeichen pro Bild in der Höhe

Bei der Zeichenanzahl in einer Zeile wird das ganze etwas komplizierter:

- Hier muss zunächst von einer Zeitspanne zur Übertragung einer Zeile von 25,4 $\mu$s ausgegangen werden
- Das SPI benötigt für die Übertragung eines Bytes 8 Takte und macht danach einen Takt Pause
- Da das SPI auf halber Frequenz läuft, wie der Prozessor, benötigt man insgesamt 18 Prozessortakte
- Die Zeitspanne für 18 Takte bei 20 MHz beträgt $\frac{1}{20MHz} * 18 = 0,9\mu$s
- $\frac{25,4\mu s}{0,9\mu s} = 28,22$ Zeichen pro Zeile

Durch diese Berechnung kommt man also auf insgesamt 840 Zeichen im Speicher, wobei wir für jedes Zeichen 1 Byte benötigen. Dadurch erhält man einen Speicherverbrauch von 0,84 kByte. Das sollte also realisierbar sein mit einem ATmega-Prozessor, der 1kByte RAM besitzt. Hier käme also schon der ATmega16 für eine Schwarz-Weiß-Lösung in Frage.

Will man 8 Farben realisieren, werden mindestens 3 Bit zusätzlich pro Pixel benötigt, um festzulegen, welche der 3 Farben beteiligt sind an der Darstellung. Um das Programm zu vereinfachen, gehe ich hier von zusätzlich einem Byte pro Zeichen aus, was dazu führt, dass sich für 8 Farben der Speicherverbrauch auf 1,68 kByte verdoppelt. Der ATmega16 reicht dafür natürlich nicht mehr, weshalb man auf größere Prozessoren, zB. den ATmega644, ausweichen muss.

## 2.2 Zeit

Zum Zeitaufwand behandelt dieser Vortrag drei Lösungsansätze. Der Ausgangspunkt ist hier, dass man für die drei RGB-Kanäle auch drei verschiedene serielle Signale haben möchte. Das heißt, man muss es entweder schaffen, ohne das SPI auszukommen, oder sich ein paar andere Tricks einfallen zu lassen.

1. Zum einen besteht die Möglichkeit, ein Programm in Assembler oder C zu schreiben, das eine serielle Ausgabe auf einem normalen Port erzeugen kann.
2. Eine andere Lösung wäre, das Problem auf die Hardwareebene zu heben und mit Schieberegistern die serielle Ausgabe zu erzeugen.
3. Die dritte Lösung besteht daraus, dass man das SPI weiterverwendet und über AND-Gatter die Farben, die man nicht haben möchte, ausblendet.

### 2.2.1 Konzept 1: serielle Ausgabe als Programmlösung

Bei der Programmlösung wird, wie oben erwähnt, versucht, auf einem Port, genauer auf drei Pins dieses Ports, die serielle Ausgabe zu generieren. Die Beschränkung auf einen Port halte ich deshalb für sinnvoll, da man ja bei einer möglichen Lösung nicht zuviel der Resourcen des Microprozessors verschwenden möchte, die man ggf. noch für andere Dinge verwenden kann. Im folgenden sieht man ein kleines Codebeispiel:

```
ausgabe2 = ausgabe2 « 1;
ausgabe3 = ausgabe3 « 2;

int j;

for (j = 0; j < 8; j++)
{
int ausgabe = (ausgabe1 & 0x01) || (ausgabe2 & 0x02) || (ausgabe3 &
0x04);
PORTD = ausgabe;
ausgabe1 = ausgabe1 » 1;
ausgabe2 = ausgabe2 » 1;
ausgabe3 = ausgabe3 » 1;
}
```

Die Variablen ausgabe1, ausgabe2 und ausgabe3 bekommen irgendwo einen Wert aus dem VideoRAM zugewiesen. Vor der Schleife müssen ausgabe2 und ausgabe3 entsprechend 1 Bit bzw. 2 Bit nach rechts geshiftet werden, damit sie korrekt auf den ersten 3 Pins von Port D ausgegeben werden können. Wenn nun die forschleife startet, erhält man an Port D in entsprechender Reihenfolge die 3 Byte seriell ausgegeben auf den ersten 3 Pins.

An dem Codebeispiel sieht man schon recht deutlich, dass die Zeit, die zur Verfügung steht, für diesen Programmablauf nicht reicht. Legt man die Schwarz-Weiß-Lösung aus technischer Informatik C zugrunde, bei der sich die Ausgabe eines Bytes innerhalb von 18 Takten abspielt, sieht man recht schnell, dass diese Zahl deutlich überschritten wird. Dieser Ansatz ist deshalb ungeeignet, zumindest solange wir wie geplant einen ATmega-Prozessor nutzen.

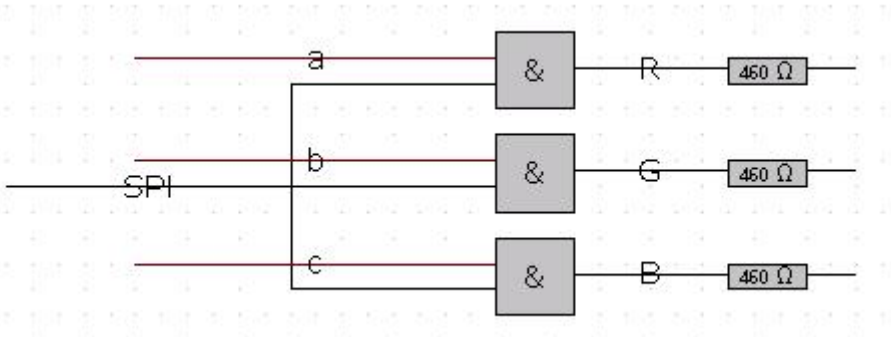
### 2.2.2 Konzept 2: serielle Ausgabe über Schieberegister

Das Schieberegister, das hier verwendet werden muss, wird digital angesteuert und erzeugt eine serielle Ausgabe. Da man pro Zeile für jedes Zeichen immer genau ein Byte übertragen möchte, braucht das Schieberegister folglich 8 Eingänge. Jetzt kann man entweder wieder 3 Ports nutzen, oder man überlegt sich ein Programm,

das über einen Port alle 3 Schieberegister ansteuert. Das wird deshalb möglich, da die Schieberegister auch einen Takt besitzen und zwischen einlesen und ausgeben umgeschaltet werden kann. Hier nimmt man 3 weitere Pins eines anderen Ports hinzu, um das Umschalten zu gewährleisten. Wenn man eine zwischengeschaltete Logik aus AND- und OR-Gattern verwendet, reichen sogar nur 2. Man legt nacheinander die entsprechenden Werte für die drei Farben an und schaltet die Schieberegister ebenfalls nacheinander auf einlesen. Ein Vorteil gegenüber SPI ist, dass man einen Takt spart, in dem das SPI Pause machen muss. Ein anderer Vorteil wäre, dass diese Hardwarelösung natürlich wesentlich weniger den Prozessor beansprucht, als es bei Konzept 1 der Fall ist. Bei dieser Lösung ist übrigens die Hintergrundfarbe ebenfalls seperat einstellbar. Ein großer Nachteil ist jedoch, dass man diesen ganzen Aufwand betreibt und trotzdem weiterhin nur 8 Farben erzeugen kann.

### 2.2.3 Konzept 3: Farben ausblenden

Beim dritten Konzept verwendet man die Ansteuerung der 3 Farbkanäle wie bei Schwarz-Weiß über SPI weiter. Jedoch wird das ganze um 3 AND-Gatter ergänzt, die zwischen das SPI und die einzelnen Farbausgänge geschaltet werden. An einem anderen Port werden nun 3 Werte abgegriffen, die das SPI entweder durchschalten oder blockieren. So kann man nicht benötigte Farben ausblenden und über diese Subtraktion an die gewünschte Farbe kommen. Die Vorwiderstände der einzelnen Farbkanäle müssen entsprechend angepasst werden. Diese Lösung bietet $2^3 = 8$ verschiedene Farben, reiht sich also in die vorhergegangen Konzepte ein. Der Unterschied besteht darin, dass sich hier die Hintergrundfarbe nicht ändern lässt. Der Vorteil dieser Lösung ist jedoch, dass sie ohne großen Mehraufwand in die bestehende Ansteuerung für Schwarz-Weiß integrieren lässt. Auch der Code lässt sich noch für diesen Zweck verändern, da an der entsprechenden Stelle noch 6 Takte zur Verfügung stehen. Diese sollten reichen, um einen Wert aus dem RAM zu laden und an einem Port auszugeben.




Ein Vorschlag für eine mögliche Kennzeichnung der 8 Farben im Speicher:

| Blau | Grün | Rot | Hex im Speicher | Farbe |
|---|---|---|---|---|
| 0 | 0 | 0 | 0x00 | Schwarz |
| 0 | 0 | 1 | 0x01 | Rot |
| 0 | 1 | 0 | 0x02 | Grün |
| 0 | 1 | 1 | 0x03 | Gelb |
| 1 | 0 | 0 | 0x04 | Blau |
| 1 | 0 | 1 | 0x05 | Magenta |
| 1 | 1 | 0 | 0x06 | Cyan |
| 1 | 1 | 1 | 0x07 | Weiß |

## 2.3 Fundstücke

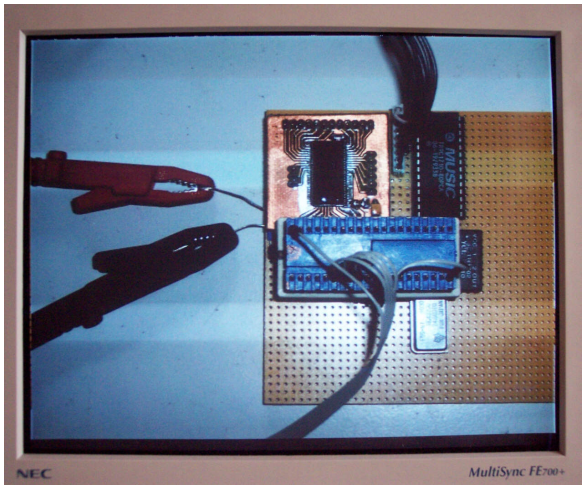
Die beiden Fundstücke beleuchten ein paar andere Ansätze zu diesem Problem, wobei es sich beim zweiten Beispiel um einen reinen Testbildgenerator handelt, der dementsprechend kaum Video-RAM benötigt.

### 2.3.1 mega8515 mit zusätzlichem SDRAM

Die folgende Lösung stammt vom User „Benedikt“ aus dem Microcontroller-Forum. Sie beinhaltet neben einem mega8515 noch zusätzliche 8MB SDRAM, die aus einem 168-poligen SDRAM DIMM entnommen wurden. Das Programm läuft mit einem Takt von 18,432MHz. Diese Version ist deshalb interessant, weil sich durch den Arbeitsspeicher viel mehr Farben und komplexere Bilder realisieren lassen, als nur mit einem Prozessor. In den folgenden Beiträgen im Forum wird jedoch auch darauf eingegangen, dass hier ebenfalls ein Textmodus sinnvoll ist, da die Erzeugung der Bilder ansonsten zuviel Zeit in Anspruch nimmt. Benedikt beziffert die Auslastung des Prozessors bei einer Bildfolge mit über 90%.
Die Realisierung hat im Bildmodus folgende Eigenschaften:

- Auflösung 512x480 Pixel
- Vertikalfrequenz 60 Hz
- 256 Farben

Im folgenden sieht man ein Beispielbild aus dem Forum, bei dem Benedikt seine Schaltung fotographiert hat und diese danach als BMP über seine Schaltung auf einem Monitor darstellt.




Der integrierte Textmodus ermöglicht eine Auflösung von 64x60 Zeichen. Hier lässt sich im Forum ebenfalls ein Beispielbild finden.

Das Projekt zeigt, dass es möglich ist, mithilfe eines Microprozessors und zusätzlichem RAM durchaus komplexere Bilder darstellen zu können, was mit Sicherheit auch eine Option für das Projekt des EinChip-Computers ist. Da man hier aber auch sehen kann, dass der Aufwand zur Bilderzeugung bei einem Prozessor enorm hoch ist, ist es möglicherweise sinnvoll, für den Computer eine eigene Grafikkarte mit einem seperaten Prozessor für die Grafikerzeugung zu verwenden.

### 2.3.2 Testbildgenerator mit ATmega8

Vom selben Nutzer aus dem Microcontroller-Forum stammt der Testbildgenerator, realisiert über einen ATmega8. Dieser ermöglicht die Darstellung eines Bilds bei 128x124 Pixeln und 16 Farben mit einer Standard-VGA Auflösung von 640x480

und dementsprechend einer Vertikalfrequenz von 60 Hz. Die merkwürdige Pixelauflösung kommt dadurch zustande, dass der Flashspeicher praktisch randvoll beschrieben wird. Benedikt schreibt im Forum, dass von den 8kByte Flashspeicher noch 36 Byte frei sind. Hier sieht man auch noch einmal sehr schön, wieviel RAM man benötigt, um ein eigentlich recht simples Bild mithilfe eines Microprozessors über VGA an einen Monitor zu übergeben.


## 3 Fazit

Die schnelle Lösung für 8 Farben wäre natürlich das dritte Konzept, da man nur verhältnismäßig wenige Änderungen am bestehenden Programmcode durchführen und 3 AND-Gatter zwischenschalten müsste. Jedoch bringt diese Lösung einige Einschränkungen mit sich, wie zB. die fehlende Hintergrundfarbe, die Einschränkung auf 8 Farben, die Begrenztheit auf einen Textmodus mit ziemlich niedriger Auflösung und so weiter. Der Ansatz von Benedikt, externen RAM mit einzubeziehen, scheint hier deutlich bessere Ergebnisse zu liefern. Weiter ausführen kann man diesen Gedanken damit, dass man auch eine Grafikkarte basteln könnte mit einem Prozessor, der sich ausschließlich um die grafischen Ausgaben kümmert. Damit hätte der Hauptprozessor deutlich mehr Zeit für andere Dinge, wie zB. andere Geräte (Joystick, Tastatur), die Soundausgabe und so weiter.

# 4 Quellen


- $http://www.serasidis.gr/circuits/AVR_VGA/avr_vga.htm$
- $http://www.mikrocontroller.net/topic/25091$
- $http://www.uni-koblenz.de/~physik/informatik/MCU/$
- $http://www.epanorama.net/documents/pc/vga_timing.html$
- $http://info.electronicwerkstatt.de/bereiche/monitortechnik/vga/Standard-Timing/index.html$
- $http://www.mikrocontroller.net/topic/25108$
- $http://en.wikipedia.org/wiki/Video_Graphics_Array$
- $http://de.wikipedia.org/wiki/Video_Graphics_Array$
- $http://www.uni-koblenz.de/~physik/informatik/techC.pdf$