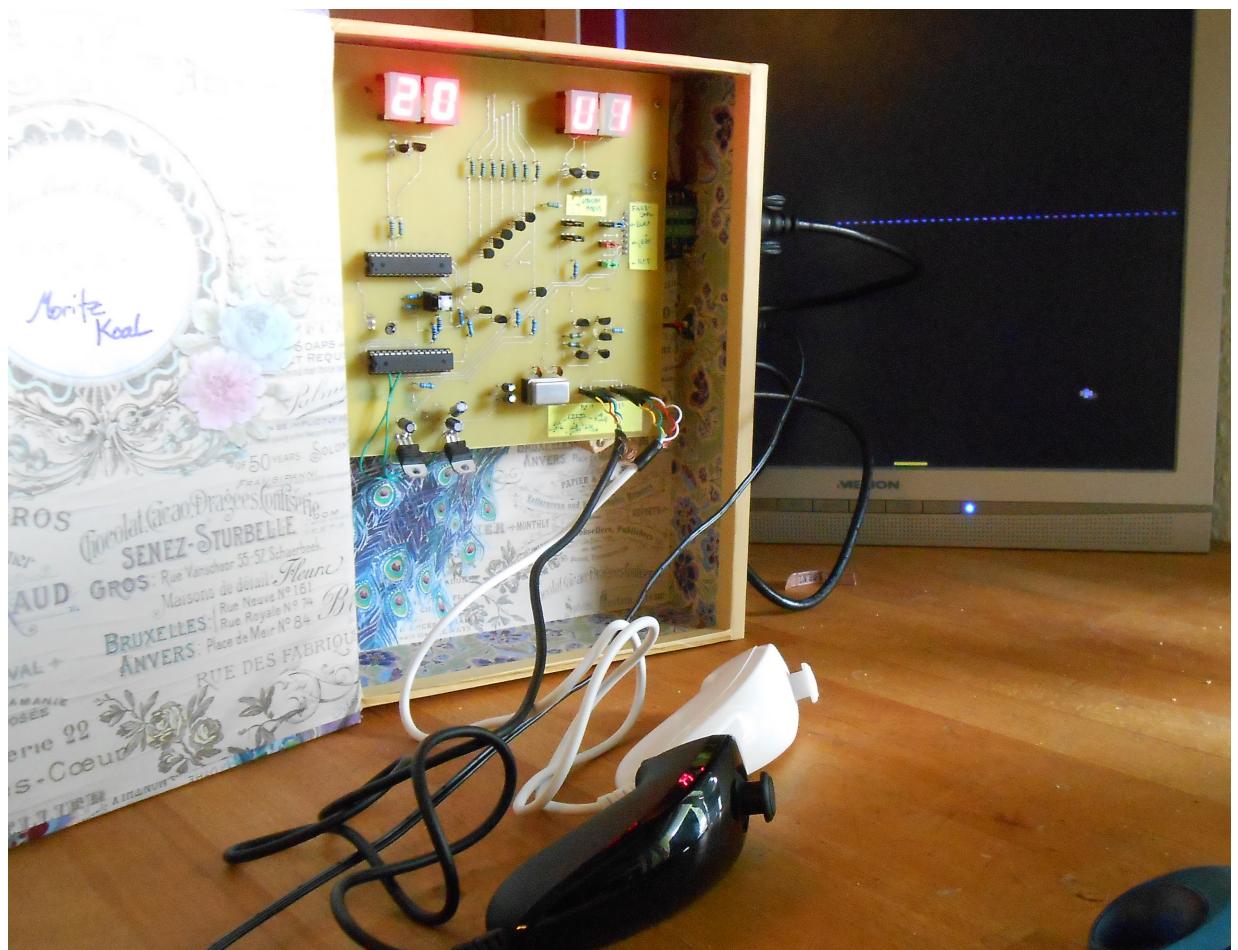


---

# Entwicklung einer simplen Spielkonsole auf Basis des AtMega88-20PU

---

von Moritz Koal



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 VGA-Modul</b>	<b>2</b>
2.1 VGA-Grundlagen . . . . .	2
2.2 VGA-Modus . . . . .	3
2.3 VGA-Umsetzung . . . . .	4
2.3.1 Notwendige Voraussetzungen . . . . .	4
2.3.2 Anmerkung zum Pixeltakt . . . . .	5
2.4 Softwaretechnischer Aufbau . . . . .	6
2.4.1 Organisation der Grafiken . . . . .	6
2.4.2 UART . . . . .	7
<b>3 Controller-Modul</b>	<b>8</b>
3.1 Ansteuerung der Nunchuks . . . . .	9
3.2 Multiplexen der Siebensegmentanzeige . . . . .	9
3.3 Resetknopf . . . . .	9
<b>4 Hardware</b>	<b>10</b>
<b>5 Anmerkung zum Platinenlayout</b>	<b>11</b>

# Abbildungsverzeichnis

1 Aufteilung des Bildschirms . . . . .	2
2 VGA-Stecker ( weiblich ) . . . . .	3
3 VGA-Modi . . . . .	4
4 Ausgaberoutine . . . . .	5
5 Softwaretechnischer Aufbau des VGA-Moduls . . . . .	7
6 Aufbau der Datagramme . . . . .	8
7 Softwaretechnischer Aufbau des Controller-Moduls . . . . .	8
8 Da müssen die Beinchen hin . . . . .	11
9 Aus den Datenblättern . . . . .	11

# 1 Einleitung

Bei dem vorliegenden Projekt handelt es sich um eine simple Spielekonsole, welche das Spiel „Pong“ auf einem angeschlossenen VGA-Monitor wiedergibt.

Als Gamecontroller dienen zwei alte Wii-Nunchucks. Ein Array von vier Siebensegmentanzeigen dient zur Ausgabe der Spielstände und über Jumper auf dem Board können zusätzliche Einstellungen ( bzgl. Farben , Steuerung über Joystick oder Beschleunigungssensoren ) getätigt werden. Darüber hinaus besitzt die Konsole einen Reset/Startknopf und drei Ausgabepins, welche als Trigger für ein simples Soundsystem dienen können<sup>1</sup>. Die sportliche Herausforderung des Projektes liegt in seiner Realisierung durch nur zwei Atmega88 20PU. Für die Kommunikation mit dem VGA-Monitor musste, aufgrund der geringen Taktfrequenz, der Spielraum des Atmega´s durch Übertaktung, Stackmanipulationen für ein simples Multithreading und geeignete Auswahl der verfügbaren Assembler-Operationen optimal ausgereizt werden.

Weniger zeitkritisch ist die Kommunikation mit den Wii-Nunchucks, die vom zweiten Atmega abgewickelt wird. Die entsprechenden Routinen wurden für Letztgenanntes komplett in C programmiert und bietet sich auch für eine Verwendung in eigenen Projekten an.

Die beiden Quellcode-Dateien, sowie der Schaltplan, Layout, Bauteilliste und einer kurzen Dokumentation liegen bei. In der Dokumentation wird kurz auf den softwaretechnischen Aufbau der beiden Atmega´s eingegangen. Diese werden im Folgenden entsprechend ihrer Aufgabe als VGA-Modul und als Controller-Modul bezeichnet.

Abschließend ist zu sagen, dass in diesem Projekt das Spielprinzip und die VGA-Ansteuerung in Assembler codiert wurde, um den technischen Limitationen zu genügen. Der dabei entstandene Assembler-Programmcode ( 2000 Zeilen ) hat dadurch nicht an Übersicht gewonnen. Wer deshalb Spiele mit komplexerer Spieldynamik schreiben will, dem sei zu Plattformen geraten, deren Taktrate eine Programmierung in C begünstigen.

---

<sup>1</sup> Letzteres ist nicht Bestandteil dieses Projektes und soll lediglich als Anregung dienen. Der ausgegebene Pegel wird nach Erzielen eines Punktes 17 ms auf HIGH ( 5 Volt ) gezogen. Es empfiehlt sich eine galvanische Trennung ( bsp. über einen Optokoppler ) von der Konsole

## 2 VGA-Modul

Das VGA-Modul übernimmt die Kommunikation mit dem Bildschirm, verwaltet die Grafiken und führt die Berechnungen der Spielmechanik durch. Darüber hinaus teilt es dem Controller-Modul Änderungen der Spielstände mit. Als Programmiersprache wurde aufgrund des zeitkritischen VGA-Aspektes „Assembler“ gewählt. Zum besseren Verständnis des softwaretechnischen Struktur wird vorab ein kurzer Einblick in die Grundlagen und Anforderungen von VGA gegeben.

### 2.1 VGA-Grundlagen

Jedes darzustellendes Bild lässt sich auf dem Bildschirm in kleinste Bildpunkte, sogenannte Pixel, unterteilen. Diese lassen sich in horizontalen Zeilen angeordnet, wobei die vertikale Summierung aller Zeilen ( folglich aller Pixel ) wieder das Gesamtbild ergibt. Ein Bildaufbau beginnt immer mit dem Pixel oben links im Bild. Bis zum Ende der

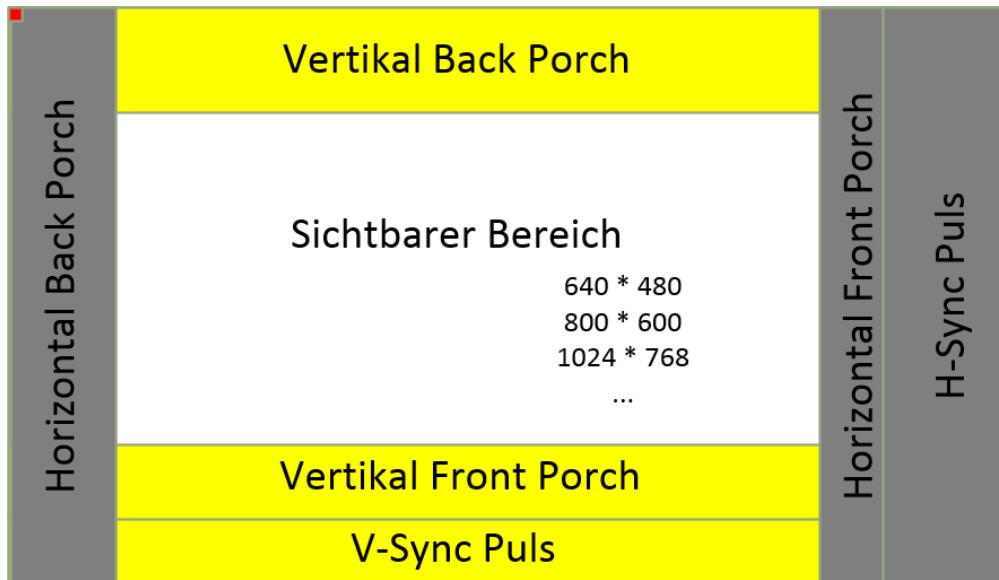


Abbildung 1: Aufteilung des Bildschirms

Zeile wird nachfolgend im Pixeltakt jeder Pixel derselben Zeile ausgewählt. Die Farbwerte des aktuell ausgewählten Pixel ergeben sich aus den Spannungspegeln, die an den drei Farbpins ( R,G,B ) anliegen. Die Intensität der Farbe ergibt sich analog in Bezug auf eine interne Referenzspannung. Eine Differenz von 1 Volt ergibt das Maximum der Intensität des jeweiligen RGB-Farbanteils. Am Ende einer Zeile erwartet der Bildschirm ein HSYNC-Signal, welches ihn veranlasst in die nächste unterliegende Zeile zu springen. Nach diesem Schema erfolgt eine Abarbeitung aller nachfolgenden Zeilen bis hin zur letzten. Am Ende dieser erwartet der Bildschirm ( neben dem HSYNC-Signal ) das VSYNC-Signal, welches durch den Sprung zurück in die erste Zeile einen erneuten Bild-

aufbau intoniert. Da mit jedem VSYNC-Signal einer neuer Bildaufbau beginnt, wird dessen Frequenz als Bildwechselfrequenz bezeichnet. Sie ergibt sich aus dem Kehrwert der Bildaufbaudauer und beträgt abhängig vom VGA-Modus ( siehe Abbildung 3 ) um die 60-75 Hz. Anzumerken sei, dass das VSYNC-Signal mehrere Zeilen anliegt, wohingegen das HSYNC-Signal immer das Ende einer Zeile markiert.

Der gesamte Bildschirm lässt sich in einen sichtbaren und einen nicht-sichtbaren Bereich aufteilen. Verdeutlich wird dies in Abbildung 1. Liegt mindestens eins der beiden Signale ( VSYNC, HSYNC ) an, werden am Bildschirm keine Pixel ausgegeben. Gleches gilt für die Zeitspanne kurz vor ( FRONT ) oder kurz nach einem Signal ( BACK ). Die Dauer der entsprechenden Zeitspanne hängt ebenfalls vom VGA-Modus ab. Den nicht-sichtbaren Bereich mit dem jede Zeile startet, wird als Schwarzschorter bezeichnet. In diesem wird die an den jeweiligen RGB-Pins anliegenden Spannung als Referenzspannung übernommen.

Als Abschluss der Grundlagen zeigt Abbildung 2 einen typischen VGA-Stecker inklusive der Pin Bezeichnung. Im Rahmen dieses Projektes werden nur die RGB-Pins, HSYNC, VSYNC und GND genutzt, wobei alle Massen zu einer kurzgeschlossen werden.

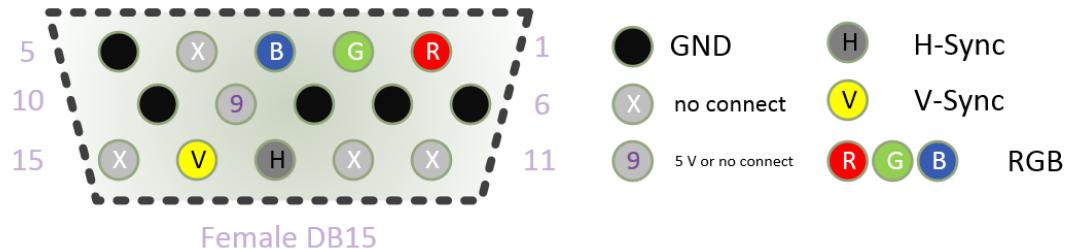


Abbildung 2: VGA-Stecker ( weiblich )

## 2.2 VGA-Modus

Indem VGA mehrere Modi anbietet, können verschiedene Auflösungen realisiert werden. Einen kurzen Überblick über einige dieser Modi bietet Abbildung 3. Die Zahlen in H-Active, H-Front, H-SYNC und H-Back entsprechen den Pixel pro Abschnitt einer Zeile ( horizontal ). Diese gelten für den in der linken Spalte angegebenen Pixeltakt. Aus dem Produkt des Kehrwertes des Pixeltaktes und den Pixel des jeweiligen Abschnittes ergibt sich die Dauer des jeweiligen Abschnittes<sup>2</sup>. Die Werte der Abschnitte V-Active, V-Front, V-SYNC und V-Back beziehen sich auf die vertikalen Abschnitte und somit jeweils auf das Vielfache einer Zeile.

---

<sup>2</sup> Als Beispiel: Bei einem Pixeltakt von 25,175 MHz werden für 640 Pixel eine Zeitdauer von  $(640/25,175) \mu\text{s}$  benötigt

Resolution	pixel clock (MHz)	Horizontal ( pixel clocks )				Vertical ( rows )			
		H-Display	H-Front-Porch	H-SYNC	H-Back-Porch	V-Display	V-Front-Porch	V-SYNC	V-Back-Porch
640x480 @60Hz	25.175	640	16	96 (low)	48	480	10	2 (low)	33
640x480 @75Hz	31.5	640	16	64 (low)	120	480	1	3 (low)	16
800x600 @72Hz	50	800	56	120 (high)	64	600	37	6 (high)	23
1024x768 @60Hz	65	1024	24	136 (low)	80	768	3	6 (low)	29
1024x768 @75Hz	78,75	1024	16	96 (high)	176	768	1	3 (high)	28

Abbildung 3: VGA-Modi

## 2.3 VGA-Umsetzung

In diesem Projekt wird der Bildschirm im 640\*480 Modus mit einer Bildwechselfrequenz von 60Hz betrieben.

640 bezeichnet die Anzahl der sichtbaren Pixel pro Zeile und 480 die sichtbaren Zeilen pro Durchgang.

Insgesamt werden in diesem Modus standardmäßig 800 Pixel pro Zeile (640+160) und 525 Zeilen (480+45) pro Bildaufbau verwendet ( siehe Abbildung 3 ). Mit einem standardmäßigen Pixeltakt von 25 MHz kommen wir dabei auf eine Zeilenfrequenz von 31,5 kHz bis 32 kHz. Dies entspricht der Frequenz des Signals HSYNC, welches den Start einer neuen Zeile einleitet.

Das gesamte Bild wird mit einer Frequenz von 60 Hz ( == 32 kHz/525 ) komplett erneuert. Dies entspricht der Frequenz des Signals VSYNC, welches den Start eines neuen Bildaufbaus einleitet. Als Plattform für die VGA Umsetzung dient ein AtMega88 20PU, der auf 22,1145 MHz übertaktet wird. Als Programmiersprache wurde für diesen auf Grund der zeitkritischen Bedingungen Assembler gewählt<sup>3</sup>.

### 2.3.1 Notwendige Voraussetzungen

Die einzigen zwingenden technischen Anforderungen des VGA-Modus beziehen sich auf das korrekte Einhalten der HSYNC und VSYNC-Frequenzen des jeweiligen Modus.

Erkennt der VGA-Monitor den Modus nicht, reagiert er mit einer Fehlermeldung. Für eine vernünftige Darstellung ist es allerdings notwendig die RGB-Pins während der Schwarzsultern auf GND zu legen, um die Referenzspannung für die nachfolgende Zeile korrekt zu definieren.

Diese zeitlichen Anforderungen definieren den jeweiligen VGA-Modus. Die Zeitspanne bzw. die Takte, die wir einen Zustand anliegen lassen müssen, ergeben sich aus Abbildung 3. Für dieses Projekt müssen diese auf die Frequenz des µC von 22.1145 MHz

---

<sup>3</sup>Das Datenblatt liefert für jeden Befehl dessen benötigte Taktzahl. So konnten taktgenaue Routinen geschrieben werden

umgerechnet werden. Durch eine Multiplikation der Takte ( hier: Pixeltakt 25 MHz ) mit  $22.1145/25$  ergibt sich die Anzahl an Takten, die jeder Abschnitt in unserem Controller anliegen muss. Die Signale HSYNC und VSYNC sind beide low aktiv und beeinflussen einander nicht ( d.h. auch wenn VSYNC aktiv ist wird trotzdem jedes Zeilenende ein HSYNC erwartet ).

### **2.3.2 Anmerkung zum Pixeltakt**

Der geforderte Pixeltakt von 25 MHz kann mit einem auf 22,1145 MHz übertakteten µC nicht eingehalten werden. Im Gegensatz zu PLD-Bausteinen, die mit einem Takt Daten laden und ausgeben können, benötigen µC's neben den Taktzyklen für die Ausgabe noch Takte zum Laden der Daten. Mit der implementierten Funktion in Abbildung 4 werden bei zwei Bit Farbinformation insgesamt vier Zyklen für jeden Pixel benötigt.

```

display_vga_data:
    // Byte |A,B|C,D|E,F|G,H| -> ausgabe jetzt in PB 3 und PB2

nop                                //+1
out FARBE_VGA_PORT,tmp              //+1  __ Bits |G,H| raus ( Alle Daten ausgegeben ! )
ld new_data,X+                      //+2  ← Hier werden neue Daten geladen
swap new_data                       //+1  __ new_data={|E,F|G,H|A,B|C,D|}_
out FARBE_VGA_PORT,new_data         //+1  __ Bits |A,B| raus
mov tmp,new_data                     //+1  __ tmp={|E,F|G,H|A,B|C,D|}_
lsl tmp                            //+1
lsl tmp                            //+1  __ tmp={|G,H|A,B|C,D|00|}_
out FARBE_VGA_PORT,tmp              //+1  __ Bits |C,D| raus
swap new_data                       //+1  __ new_data={|A,B|C,D|E,F|G,H|}_
swap tmp                            //+1  __ tmp={|C,D|00|G,H|A,B|}_
nop
out FARBE_VGA_PORT,new_data         //+1  __ Bits |E,F| raus
rjmp display_vga_data               //+2

```

Abbildung 4: Ausgaberoutine

Der tatsächliche Pixeltakt des µC entspricht damit 5.5286 MHz<sup>4</sup>.

Indem der Bildschirm die Pixel mit einem Takt von 25 MHz einzieht, sie aber nur mit einem Takt von 5.5286 geändert werden, kommt es zu einer seitlichen Streckung der Pixel<sup>5</sup>. Für den VGA-Bildschirm ist es irrelevant mit welcher Frequenz sich die Pixel am Eingang geändert werden.

Der Nachteil eines geringen Pixeltakt liegt darin, dass weniger Daten vom µC auf dem Bildschirm ausgegeben werden können. Eine Zeile fasst beispielsweise 640 sichtbare Pixel. Gäbe der µC mit einer Frequenz von 1 MHz Pixel aus, dann entspräche einem ausgegebenen Pixel des µC ungefähr 25 Pixel ( Bei den benannten 25 MHz Pixeltakt des Bildschirms ) auf dem Bildschirm. Das bedeutet nicht nur, dass die Pixel des µC

<sup>4</sup>Um diesen zu erreichen musste ein komplettes GPIO-Register genutzt werden, von dem nur zwei der acht Pins tatsächlich den Output lieferten.

<sup>5</sup>der Bildschirm nimmt beispielsweise 4-5 mal den gleichen Wert für einen Pixel, da wir diesen nur „langsam“ ( mit 5.5286 MHz ) ändern können

enorm gestreckt werden würden, sondern auch, dass der µC nur 25 Pixel darstellen kann bevor der nicht sichtbare Bereich beginnt.

Darüber hinaus entspricht die feinste Änderung einer Grafik, welche der µC erzeugen kann, einer Verschiebung um 25 Pixel auf dem Bildschirm.

## 2.4 Softwaretechnischer Aufbau

Neben der Initialisierung der einzelnen Register werden in der Startphase sämtliche zur Darstellung benötigten Daten aus dem FLASH-Speicher in den SRAM geladen.

Grund hierfür sind die wesentlich schnelleren Speicherzugriffe auf den SRAM<sup>6</sup>. Nach der Startphase werden die Pixel in der Routine „display\_vga\_data“ mit 5.5286 MHz ausgegeben ( siehe Abbildung 4 ).

Alle 31,5 µs ( dies entspricht der Frequenz von HSync, 31,7 kHz ) wird ein Timer-Interrupt ausgeführt.

Die darauf folgende ISR setzt die Spannung an den RGB-Pins auf GND ( Schwarzschorter ) und aktiviert nach neun Takten ( Dauer des FRONT-PORCH ) das Signal HSYNC. Die Zeitdauer bis HSYNC wieder deaktiviert wird ist zeilenunabhängig und nur durch den VGA-Modus festgelegt. Die Operationen, welche in dieser Zeitspanne ausgeführt werden, sind hingegen zeilenabhängig. Befindet sich die nächste Zeile beispielsweise im sichtbaren vertikalen Bereich, wird der Zeiger mit der Adresse, der als nächstes darzustellenden Zeile, geladen. Wichtig ist allerdings, dass unabhängig vom eingeschlagen Weg immer die gleiche Anzahl an Takten bzw. die gleiche Zeit benötigt wird.

Während den nicht-sichtbaren Zeilen ( VSYNC\_FRONT, VSYNC\_AKTIV, VSYNC\_BACK ) müssen keine Pixel ausgegeben werden. Folglich können in diesem Zeitraum andere Aufgaben bewältigt werden. Im Projekt findet sich dieser Bereich unter „main“. Er umfasst die Kommunikation mit dem Controller-Modul, sowie die Auswertung der Bewegungen und das Neuanlegen der darzustellenden Zeilen. Wie in Abbildung 5 dargestellt, können „main“ und „display\_vga\_data“ auf höhere Abstraktionsebene als zwei Prozesse angesehen werden. Die Timer ISR bildet in diesem Fall den Prozess-Scheduler, der anhand der Zeilenzahl entscheidet, welcher Prozess aktuell laufen darf. Über eine zusätzliche Manipulation des Stackpointer wird erreicht, dass der Prozess „display\_vga\_data“ immer am gleichen Einstiegsplatz beginnt, wohingegen „main“ an der Stelle fortgesetzt wird, an der er unterbrochen wurde. Eine Ausnahme bildet der Fall, dass „main“ neu beginnen soll d.h. die aktuelle Zeile der ersten nicht-sichtbaren Zeile entspricht. In diesem Fall wird am Start der Routine „main“ eingestiegen.

### 2.4.1 Organisation der Grafiken

Wie eingangs erwähnt, wurden sämtliche Grafiken in der Startphase im SRAM abgelegt. Die gesamte Darstellung besteht aus fünf kompletten darzustellenden Zeilen, welche im Projekt als „worklines“ bezeichnet werden. Diese werden abhängig von der aktuellen Zeile ausgewählt und auf den Bildschirm gezeichnet. Da gewisse Objekte ( Spielball, Spielerbalken ) einer ständigen Änderung unterworfen sind, müsste die zugehörigen „worklines“

---

<sup>6</sup>Vergleich: SRAM r/w = 2 Takte, FLASH r/w = 3 Takte, EEPROM r/w > 10 Takte

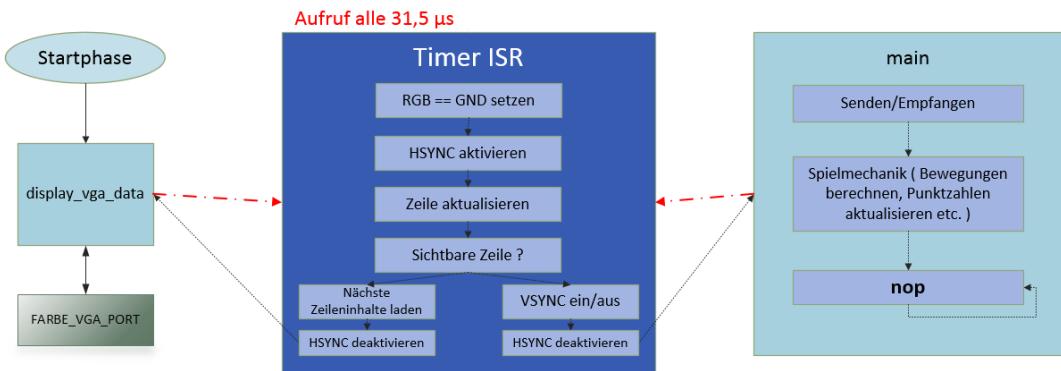


Abbildung 5: Softwaretechnischer Aufbau des VGA-Moduls

sich ebenfalls ändern können. Dafür wurden sogenannte IMAGES generiert, welche nur die jeweiligen Spielobjekte beinhalten. Diese werden jeweils an die richtigen Stellen in der zugehörigen „workline“ kopiert. Die Verarbeitung findet dabei in „main“ statt, sodass beim Bildaufbau nur noch die richtige „workline“ ausgewählt werden muss. Indem mit 2-Bit Farbwerten gearbeitet wird, entspricht die geringste Schrittweite einer Verschiebung des IMAGES innerhalb der „workline“ zwei Bit nach links oder rechts. Allerdings kann nur byteweise ( also alle 8 Bit ) adressiert werden. Durch Routinen, die es erlauben durch mehrere benachbarte Speicherbereiche zu schieben, wurde diese Problematik gelöst.

#### 2.4.2 UART

Die Kommunikation mit dem Controller-Modul wird über die serielle Schnittstelle des AtMega88 gewährleistet ( 8 Datenbits, 1 Stopbit, 0 Paritätsbits , Baudrate : 32,143 Kbps ). Die Datagramme besitzen den in Abbildung 6 dargestellten Aufbau.

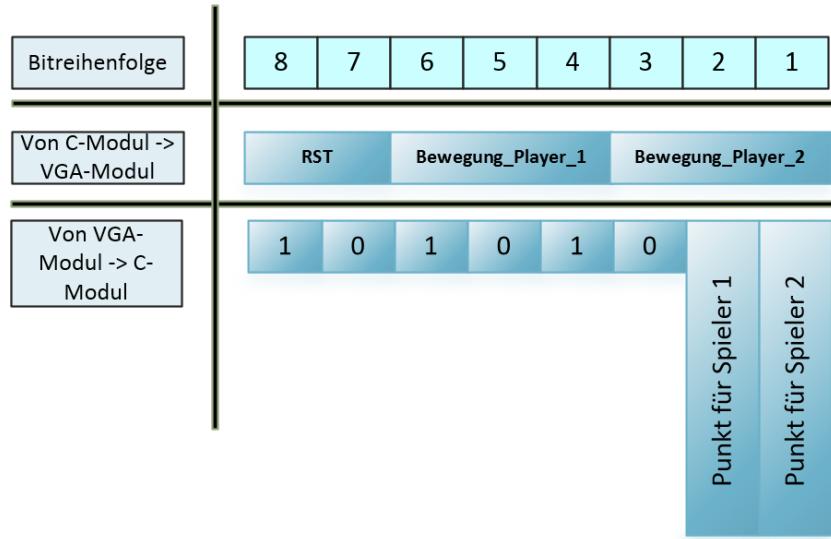


Abbildung 6: Aufbau der Datagramme

### 3 Controller-Modul

Das Controller-Modul übernimmt die Abfrage der Nunchucks, die Verwaltung der Spielstände und deren Ausgabe über die Siebensegmentanzeigen sowie den RST-Knopf und die Kommunikation mit dem VGA-Modul. Als Programmiersprache wurde C gewählt. Einen Überblick des softwaretechnischen Aufbaus liefert Abbildung 7. Nach der Initia-

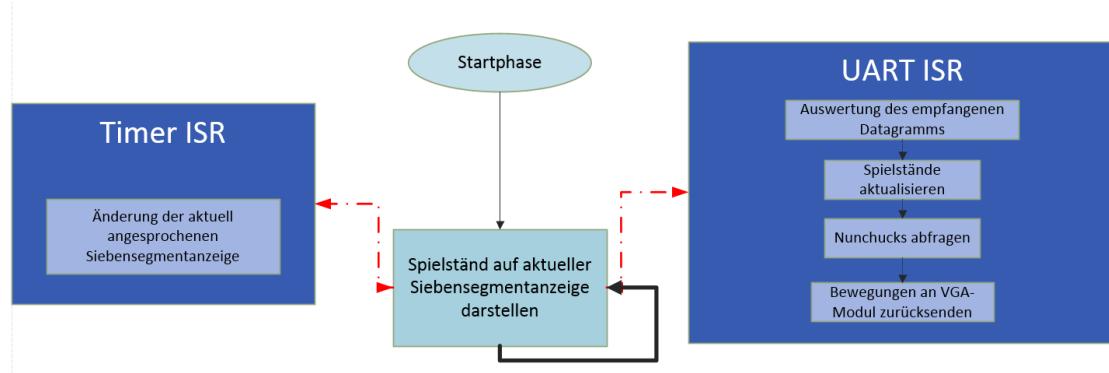


Abbildung 7: Softwaretechnischer Aufbau des Controller-Moduls

lisierung läuft das Modul in einer Endlosschleife in der es die aktuellen Werte auf der Siebensegmentanzeige ausgibt. Unterbrochen wird diese entweder durch eine Änderung der aktuell angesprochenen Siebensegmentanzeige ( Timer ISR ) oder durch ein Eintreffen der Daten des VGA-Moduls ( UART ISR ). In der ISR des UART werden die Spielstände aktualisiert, die Nunchucks abgefragt und deren Bewegungen zurück an das

VGA-Modul gesendet.

### **3.1 Ansteuerung der Nunchuks**

Der Wii-Nunchuk besitzt einen Joystick ( X, Y ), drei Beschleunigungssensoren ( X, Y, Z ) sowie zwei Taster ( C, Z ). Für dieses Projekt wird der dargestellte Balken des jeweiligen Spielers entweder über den Joystick ( X ) oder über den Beschleunigungssensor ( X ) gelenkt. Die Auswahl hierfür wird über das Setzen des jeweiligen Jumpers getroffen. Eine Kommunikation mit dem Nunchuk lässt sich über den TWI-Bus ( auch I<sup>2</sup>-Bus genannt ) realisieren. Zugewiesen wird dem Nunchuck dabei die Adresse 0x52. Zu Beginn der Nutzung steht die Initialisierungsroutine „nunchuck\_init“. Die Abholung der Daten wird anschließend über „nunchuck\_receiveData“ realisiert. Dabei sendet der Nunchuck sieben Byte mit sämtlichen Informationen ( Joystick, Beschleunigung, Taster ) zurück. Sämtliche Routinen und Initialisierungswörter können bei Bedarf dem beigefügten Quellcode entnommen werden. Auf eine ausführliche Erläuterung wird an dieser Stelle verzichtet. Anzumerken ist noch, dass beide Nunchucks an der selben TWI-Schnittstelle hängen und die selbe I<sup>2</sup>-Bus Adresse besitzen. Indem das SCL-Signal gemultiplext wird, können trotzdem beiden einzeln angesprochen werden. Hierbei muss nur die SCL-Datenleitung gemultiplext werden. Laut Erfahrungsberichten einschlägiger Foren verträgt der Nunchuck dauerhaft keine 5 Volt. Deswegen wurde ein bidirektionaler Spannungswandler zwischengeschaltet. Als zeitliches Kriterium kommt hinzu, dass die Abfrage beider Nunchucks und die Kommunikation mit dem VGA-Modul innerhalb eines Bildaufbaues geleistet werden muss. Als Frequenz für den TWI-Buses wurde daher 100kHz gewählt. Die Kabelzuordnung des originalen Nintendo-Nunchucks lautet: Gelb-> SCL , Grün-> SDA, Rot-> 3,3V Weiß-> GND.

### **3.2 Multiplexen der Siebensegmentanzeige**

Zur Umrechnung der Spielstände dient eine LookUp-Tabelle, die den Zahlen eine äquivalente Siebensegmentdarstellung zuordnet. Änderungen der Spielstände werden vom VGA-Modul empfangen. Für den Aufbau der Datagramme wird auf Abbildung 6 verwiesen. Alle vier Siebensegmentanzeigen werden mit einer Frequenz von 150 Hz gemultiplext.

### **3.3 Resetknopf**

Im Falle eines Resets werden die Spielstände genullt und im zu sendenen Datagramm die Bits 8,7 auf 0b11 gesetzt. Das VGA-Modul setzt die Darstellung anschließend in den Startzustand.

## 4 Hardware

Ein Großteil des Aufbaus kann entweder dem Datenblatt des AtMega88 entnommen werden oder/und ist durch ausreichend Tutorials im Internet bereits erläutert. Auf diese Schaltungen wird daher nicht näher eingegangen.

Im Falle des bidirektionalen Levelshifters wird auf die APPLICATION NOTE AN97055 von Philips verwiesen.

Die Schaltung des Multiplexer der Nunchucks ergibt sich bei Betrachtung des Ersatzschaltbildes eines FETs ( parallel geschaltete Diode ).

## 5 Anmerkung zum Platinenlayout

Die Markierung/ Umriss der Transistoren im Eagle-Layout entspricht nicht(!) deren Beinchenbelegung. In Abbildung 8 ist explizit markiert, welcher Anschluss für welches Beinchen gedacht ist. Mit einen Blick in die mitgelieferten Datenblätter für den BS108

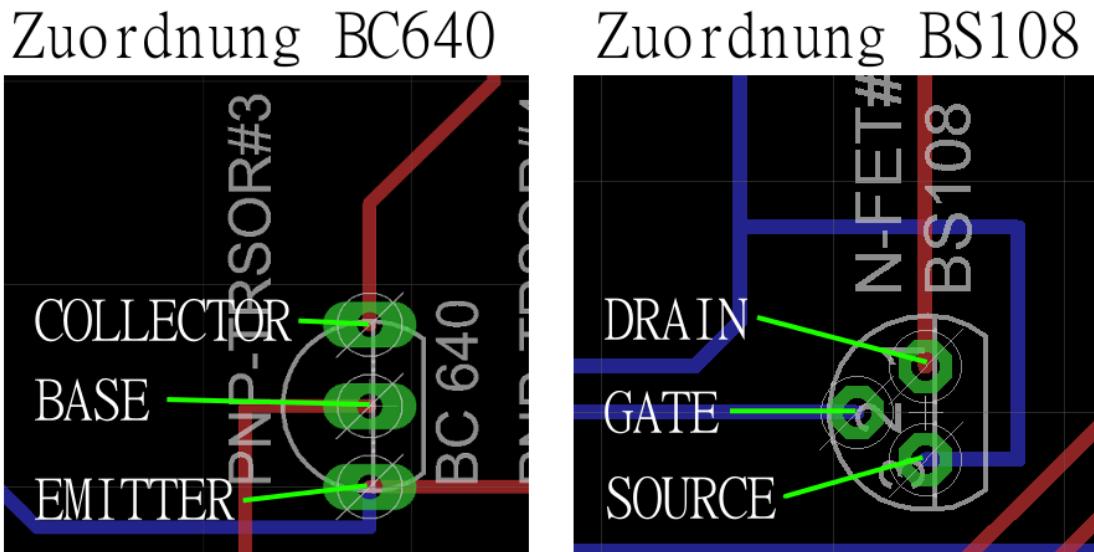


Abbildung 8: Da müssen die Beinchen hin

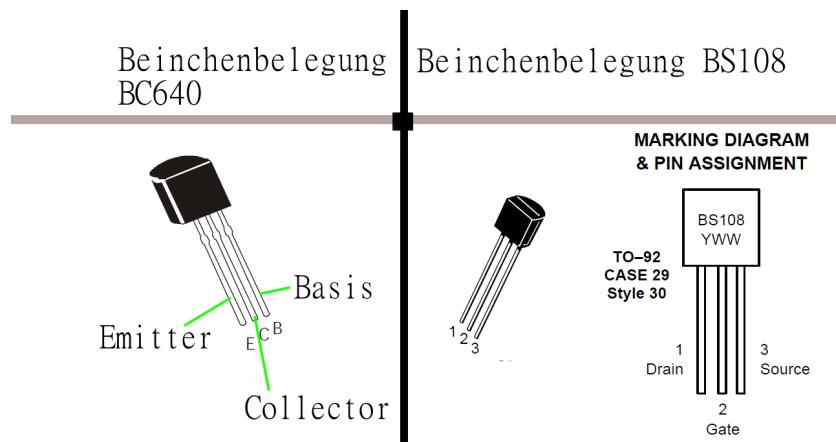


Abbildung 9: Aus den Datenblättern

und den BC640 lassen sich diese korrekt montieren. Beim BC640 müssen die Beinchen für Kollektor und Basis gekreuzt werden. Beim BS108 müssen die Beinchen von Drain und Source getauscht werden, was einer seitlichen Drehung von 180° entspricht. Ich bitte dies zu entschuldigen.