

Mikroprozessor Workshop  
Wintersemester 2019/2020

# **Vier Gewinnt auf dem Motorola 68HC11 Prozessor**

**Benutzer- und Programmierhandbuch**

Michael Persiehl (tinf102296)  
Guillaume Fournier-Mayer (tinf101922)

15. April 2020, Hamburg

## Inhaltsverzeichnis

<b>1 Benutzerhandbuch</b>	<b>2</b>
1.1 Aufgabenstellung . . . . .	2
1.2 Spielregeln . . . . .	2
1.3 Bedienung . . . . .	3
1.4 Ausgabe . . . . .	4
1.4.1 Spielstart . . . . .	4
1.4.2 Spielende . . . . .	4
1.5 Inbetriebnahme . . . . .	4
<b>2 Programmiererhandbuch</b>	<b>8</b>
2.1 Spielfeld . . . . .	8
2.1.1 Zelle . . . . .	8
2.1.2 Buffer . . . . .	11
2.2 Eingabe . . . . .	12
2.2.1 Tastenbyte auslesen . . . . .	12
2.2.2 Flankenerkennung und Entprellung . . . . .	12
2.3 Ausgabe . . . . .	14
2.3.1 LCD . . . . .	14
2.3.2 Spielfeld . . . . .	14
2.3.3 Text . . . . .	14
2.4 Cursor . . . . .	16
2.4.1 Aufbau . . . . .	16
2.4.2 Steuerung . . . . .	16
2.5 Logik . . . . .	18
2.5.1 Umrechnung der Boardadresse in eine Bufferadresse . .	18
2.5.2 Prüfen des Zelleninhalts . . . . .	18
2.5.3 Spielerwechsel . . . . .	19
2.5.4 Feststellen der Anzahl an zusammenhängenden Steinen	19
<b>3 Listening</b>	<b>21</b>

# 1 Benutzerhandbuch

## 1.1 Aufgabenstellung

Für den Mikroprozessor-Workshop im Wintersemester 2020 wurde das Spiel *4 Gewinnt* auf dem Motorola 68HC11 Prozessor umgesetzt. Als Software wurde *Geany* und als Compiler wurde *miniIDE* unter Windows verwendet.

Unser Ziel ist es *4 Gewinnt* für zwei Spieler auf dem 68HC11 Board in Assembler zu programmieren. Für die Steuerung werden dabei die Tasten-, zur Anzeige der Spieldaten der LCD des Boards genutzt.

## 1.2 Spielregeln

Das Spiel wird auf einem 7 Felder breiten und 6 Felder hohen Spielbrett gespielt (Abbildung ??), in das die Spieler abwechselnd ihre Spielsteine fallen lassen. Jeder Spieler besitzt 21 gleichfarbige Spielsteine (Spieler 1 komplett gefüllt, Spieler 2 innen leer. (Abbildung ??))

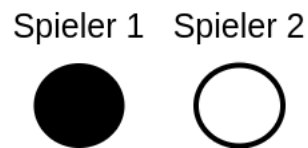


Abbildung 1.1: Spieler

Wenn ein Spieler einen Spielstein in eine Spalte fallen lässt, besetzt dieser den untersten freien Platz der Spalte. Gewinner ist der Spieler, der es als erster schafft, vier oder mehr seiner Spielsteine waagerecht, senkrecht oder diagonal in eine Linie zu bringen. Das Spiel endet unentschieden, wenn das Spielbrett komplett gefüllt ist, ohne dass ein Spieler eine Viererlinie gebildet hat.

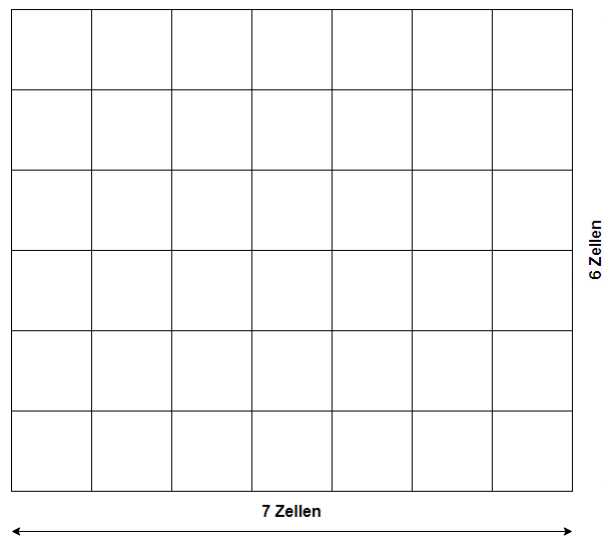


Abbildung 1.2: Spielfeld

### 1.3 Bedienung

Das Spiel wird mit vier Tasten gesteuert. Dabei wird der Cursor mit der linken und der rechten Taste, jeweils um eine eine Spalte in die gewünschte Richtung bewegt. Mit der unteren Taste wird ein Stein, an der aktuellen Cursorposition, in die Spalte geworfen. Über die Resettaste kann das komplette Spiel zurückgesetzt werden. Spieler 1 ist nun wieder am Zug.

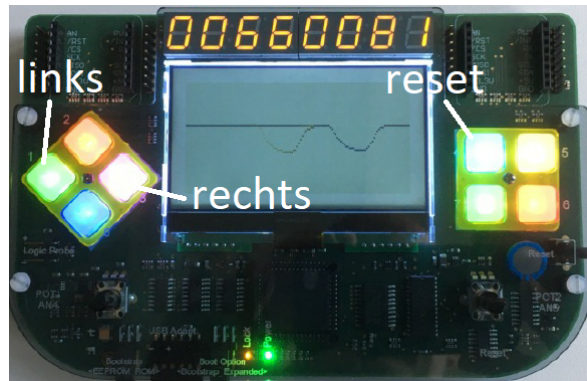


Abbildung 1.3: Spielfeld

## 1.4 Ausgabe

### 1.4.1 Spielstart

Nach erfolgreichem Programmstart wird das leere Spielfeld in der mitte des LCD's angezeigt. Links davon befindet sich ein Textfeld *turn*, welches den Namen des Spielers anzeigt, der zurzeit dran ist. Beim Programmstart sowie beim Zurücksetzen des Spieles startet immer *Spieler 1*. Unter dem Spielfeld ist der Cursor zu sehen, der zum Start des Spieles auf die mittlere Spalte zeigt.

### 1.4.2 Spielende

Falls ein Spieler gewonnen hat, wird unter dem Spielfeld der gewinnende Spieler angezeigt. Zusätzlich ist Eingabe blockiert und das Spiel lässt sich nur mit den Resetbutton zurücksetzen.

## 1.5 Inbetriebnahme

Zum Ausführen des Spiels muss zuerst die Assemblerdatei in einer IDE wie z.B. Geany (Freeware: <https://www.geany.org/>) geöffnet werden (rote Markierung, Abbildung ??).

## 1 Benutzerhandbuch

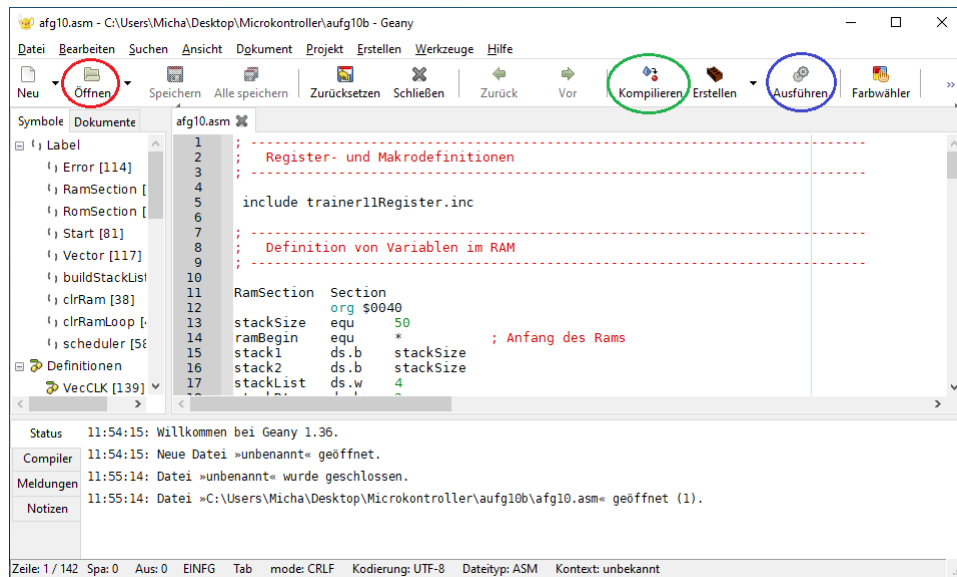


Abbildung 1.4: Zellenbelegung

Als nächstes müssen die Einstellungen in Geany geändert werden: Zunächst der verwendete Compiler, welchen wir von miniIDE nutzen, welche vorher im Verzeichnis `C:\miniide` installiert sein muss (ansonsten muss der Pfad zum Compiler später entsprechend angepasst werden). Ebenfalls benötigt wird die Software *Realterm* in dem Verzeichnis `C:\realterm`. Unter dem Stichwort *Dokument* → *Dateityp festlegen* → *Kompilersprachen* muss *Assembler* ausgewählt werden, danach im Menü-band unter *Erstellen* erreicht man den Punkt *Kommandos zum Erstellen konfigurieren* (Abbildung ??).

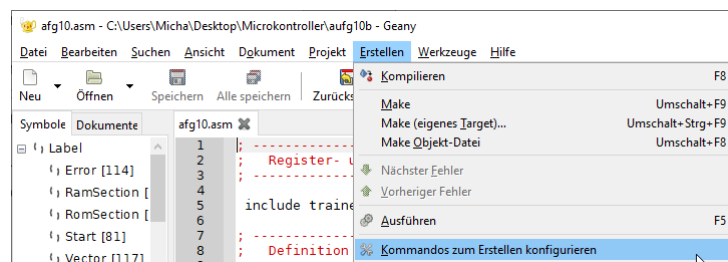


Abbildung 1.5: Geany Konfiguration

## 1 Benutzerhandbuch

Hier muss nun unter *Kommandos für ASM* folgendes eingetragen werden:

```
C:\miniIDE\ASM11 %f -l
```

und unter *Befehle zum Ausführen*:

```
C:\Realterm\realterm "first" "display=1" "rows=40" "baud=9600"  
"sendfile=%d\%e.s19"
```

#	Label	Kommando	Arbeitsverzeichnis	Zurücksetzen
<b>Kommandos für ASM</b>				
1.	Assembler	C:\miniIDE\ASM11 %f -l		
2.				
3.				
Regulärer Ausdruck für Fehlermeldungen:				
<b>Dateityp-unabhängige Befehle</b>				
1.	Make	make		
2.	Make (eigenes Target)...	make		
3.	Make Objekt-Datei	make %e.o		
4.				
Regulärer Ausdruck für Fehlermeldungen:				
Notiz: Element 2 öffnet ein Dialog und fügt das Ergebnis am Ende des Kommandos an				
<b>Befehle zum Ausführen</b>				
1.	Ausführen	C:\Realterm\realterm "first" "display=1" "rows=40" "baud=9600" "sendfile=%d\%e.s19"		
2.				

%d, %e, %f, %p, %l werden innerhalb der Kommando- und Verzeichnissfelder ersetzt - Details gibt es in der Dokumentation.

Abbrechen OK

Abbildung 1.6: Geany Konfiguration

Gegebenenfalls muss hier noch *Port=* mit der Nummer des entsprechenden COM-Ports ergänzt werden.

Nach dem bestätigen per *OK* müssen die Daten kompiliert und auf das Board überspielt werden (grüne und blaue Markierungen, Abbildung ??). Nach dem Klick auf *Ausführen* öffnet sich jetzt *Realterm* (Abbildung ??). Nach betätigung des Resetschalters können die Daten mit *Send File*, im Reiter *Send*, ans Board übertragen werden.

## 7

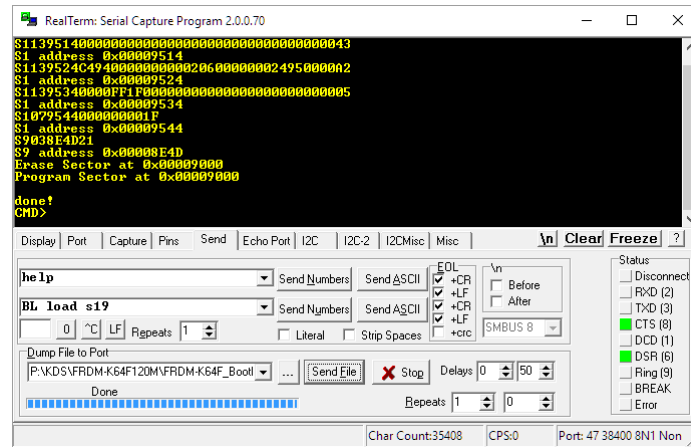


Abbildung 1.7: Realterm



## 2 Programmiererhandbuch

### 2.1 Spielfeld

In dem folgenden Kapitel wird das Spielfeld, dessen Aufbau in Zellen und die interne Repräsentation, als Buffer, im Speicher erläutert.

#### 2.1.1 Zelle

Eine Zelle wird auf dem LCD als Block von Pixeln betrachtet. Dabei besteht die Zelle aus folgender Formel:

$$64Pixel = 8Pixel \cdot 8Pixel \quad (2.1)$$

Diese 64 Pixel werden intern als acht hintereinander liegende Bytes repräsentiert. Dabei steht das erste Bit des ersten Bytes für den Pixel in der oberen linken Ecke. Um ein Pixel anzusteuern, wird das jeweilige Bit auf 1 bzw. auf 0 gesetzt.

	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte	8. Byte
1. Bit	1	1	1	1	1	1	1	1
2. Bit	2	2	2	2	2	2	2	2
3. Bit	3	3	3	3	3	3	3	3
4. Bit	4	4	4	4	4	4	4	4
5. Bit	5	5	5	5	5	5	5	5
6. Bit	6	6	6	6	6	6	6	6
7. Bit	7	7	7	7	7	7	7	7
8. Bit	8	8	8	8	8	8	8	8

Abbildung 2.1: Darstellung einer Zelle im RAM

### Leere Zelle

Eine Leere Zelle ist jene, die kein Spielstein beinhaltet und somit nur aus Rand besteht. Um den vertikalen Rand darzustellen, müssen alle Bits des ersten und achten Bytes auf 1 gesetzt werden. Für den horizontalen Rand müssen alle ersten und achten Bits des 2,3,4,5,6 und 7 Bytes auf 1 gesetzt werden.

	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte	8. Byte
1. Bit	1	1	1	1	1	1	1	1
2. Bit	2	2	2	2	2	2	2	2
3. Bit	3	3	3	3	3	3	3	3
4. Bit	4	4	4	4	4	4	4	4
5. Bit	5	5	5	5	5	5	5	5
6. Bit	6	6	6	6	6	6	6	6
7. Bit	7	7	7	7	7	7	7	7
8. Bit	8	8	8	8	8	8	8	8

Abbildung 2.2: Darstellung einer leeren Zelle im RAM

### Spieler 1 Zelle

Eine Zelle mit einem Spielstein von Spieler 1 ist jene, die aus Rand und aus einem gefüllten Spielstein besteht.

## 2 Programmiererhandbuch

	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte	8. Byte
1. Bit	1	1	1	1	1	1	1	1
2. Bit	2	2	2	2	2	2	2	2
3. Bit	3	3	3	3	3	3	3	3
4. Bit	4	4	4	4	4	4	4	4
5. Bit	5	5	5	5	5	5	5	5
6. Bit	6	6	6	6	6	6	6	6
7. Bit	7	7	7	7	7	7	7	7
8. Bit	8	8	8	8	8	8	8	8

Abbildung 2.3: Darstellung einer Zelle mit einem Spielstein von Spieler 1 im RAM

### Spieler 2 Zelle

Eine Zelle mit einem Spielstein von Spieler 1 ist jene, die aus Rand und aus einem leeren Spielstein besteht.

## 2 Programmiererhandbuch

	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte	8. Byte
1. Bit	1	1	1	1	1	1	1	1
2. Bit	2	2	2	2	2	2	2	2
3. Bit	3	3	3	3	3	3	3	3
4. Bit	4	4	4	4	4	4	4	4
5. Bit	5	5	5	5	5	5	5	5
6. Bit	6	6	6	6	6	6	6	6
7. Bit	7	7	7	7	7	7	7	7
8. Bit	8	8	8	8	8	8	8	8

Abbildung 2.4: Darstellung einer Zelle mit einem Spielstein von Spieler 2 im RAM

### 2.1.2 Buffer

Das gesamte Spielfeld wird intern als Buffer repräsentiert. Änderungen am Spielfeld werden zunächst im Buffer getätigt, bevor der gesamte Inhalt an den LCD geschickt wird.

Die Größe des Buffers berechnet sich dabei aus folgender Formel:

$$\text{Buffergrösse} = \text{Zeilen} \cdot \text{Spalten} \cdot \text{Zellengrösse} \quad (2.2)$$

Da das Spielfeld aus sechs vertikalen Zellen und sieben horizontalen Zellen besteht und diese wiederum aus acht Bytes bestehen, ergibt sich folgende Buffergröße:

$$336\text{Byte} = 6 \cdot 7 \cdot 8\text{Byte} \quad (2.3)$$

## 2.2 Eingabe

In dem folgenden Kapitel wird die Eingabe durch Tastendruck, deren Entprellung und Flankenerkennung erläutert.

### 2.2.1 Tastenbyte auslesen

Um auf einen Tastendruck zu reagieren wird in regelmäßigen Abstand das *PIO\_B*-Byte ausgelesen. Dabei ist dieses *n-aus-8-Kodiert*. Jedes Bit repräsentiert dabei den Zustand eines Tasters. Ist ein Bit auf 0 gesetzt, ist die Taste zurzeit gedrückt und umgekehrt.

**Taste 0 (11111110)** Setzt abhängig davon wer zurzeit dran ist, einen entsprechenden Stein an der Cursorposition. Sobald der Stein gesetzt worden ist, wird die Logik angesteuert um einen möglichen Sieg zu ermitteln.

**Taste 1 (11111101)** Bewegt den Cursor nach Links.

**Taste 3 (11110111)** Bewegt den Cursor nach Rechts.

**Taste 4 (11101111)** Setzt das Spiel zurück.

### 2.2.2 Flankenerkennung und Entprellung

Da das einlesen des *PIO\_B*-Bytes in einer Schleife **\*\*SIEHE MAINLOOP\*\*** ausgeführt wird, muss sichergestellt werden, dass nur eine Flanke pro Tastendruck ausgewertet wird. Zusätzlich muss, durch die fehlende Hardwareentprellung der Tasten, die Entprellung in Software realisiert werden.

Dazu wird zunächst das *buttonFlag* getestet. Ist es nicht gesetzt, kann auf eine Taste reagiert und das Flag gesetzt werden. Ist es jedoch gesetzt, wird ein Timer inkrementiert. Ist dieser größer als 250, wird das *buttonFlag* zurück gesetzt, welches es wieder ermöglicht auf einen Tastendruck zu reagieren. Falls der Timer jedoch kleiner als 250 ist, muss weiterhin gewartet werden und ein Entprellen der Tasten zu gewährleisten.

## 2 Programmiererhandbuch

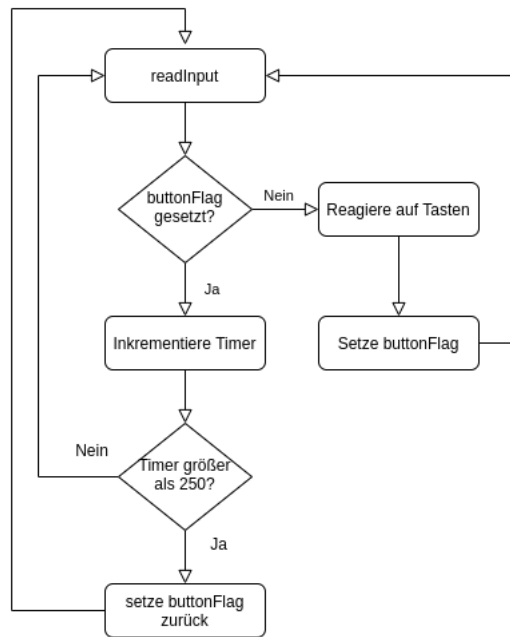


Abbildung 2.5: Programmablaufplan: *readInput*

## 2.3 Ausgabe

In diesem Kapitel wird erläutert wie die Ausgabe auf dem LCD realisiert wurde. Dabei werden die Eigenschaften des Displays und die Representation von Text innerhalb des Speichers erklärt.

### 2.3.1 LCD

Als Display steht dem Board ein *NHD-C12864A1Z-FSW-FBW-HTT* zur Verfügung. Dieses besitzt 128 x 64 Pixel. Da der Cursor jedoch 8 Pixel hoch ist, bildet das LCD 128 Spalten und acht Zeilen ab.

### 2.3.2 Spielfeld

Um das Spielfeld zu zentrieren wird auf jede horizontale Berechnung die Konstante *boardOffset* addiert.

Diese berechnet sich aus folgender Formel:

$$\text{boardOffset} = \frac{\text{Displaybreite} - \text{Spielfeldbreite}}{2} \quad (2.4)$$

Konkret bedeutet das:

$$36 = \frac{128\text{Pixel} - 7\text{Zeilen} \cdot 8\text{Pixel}}{2} \quad (2.5)$$

### 2.3.3 Text

Da kein dynamischer Text für die Ausgabe benötigt wird, wird jeglicher Text aus dem Speicher ausgelesen.

Dabei besteht ein Buchstabe aus zwei bis vier Bytes. Zusätzlich wird ein Leerzeichen, in Form eines leeren Bytes, an das Ende des Buchstaben angefügt.

Somit ergibt sich beispielsweise folgende Bytefolge für den Buchstaben *T*:

\$02,\$7E,\$02,\$00

Um nun eine Buchstabenfolge als Text auszugeben, wird die Länge dieser im ersten Byte gespeichert. Somit ist es möglich, in einer Schleife eine Buchstabenfolge auszugeben, ohne den dahinter liegenden Speicher mit auszulesen. Alternativ könnte hier auch ein Stopbyte verwendet werden. Dies hätte den Vorteil, dass auch Texte, die länger als 254 Zeichen lang sind, Ausgegeben werden können.

Die folgende Tabelle zeigt die Kodierung mit dem Beispieltext *Player*:

## 2 Programmiererhandbuch

P:	30,\$7E,\$12,\$12,\$0C,\$00
l:	\$02,\$7E,\$00
a:	\$20,\$54,\$54,\$78,\$00
y:	\$1C,\$A0,\$A0,\$7C,\$00
e:	\$38,\$54,\$54,\$48,\$00
r:	\$7C,\$08,\$04,\$08,\$00
:	\$00,\$00



## 2.4 Cursor

In diesem Kapitel wird der Cursor mit allen Funktionen und Bestandteilen erläutert. Darunter fallen unter anderem eine Variable (cursorColumn, deklariert in Viergewinnt.asm, Größe 1 Byte) zur Beschreibung der horizontalen-Position auf dem Board, welche die Spalte, die durch den Cursor ausgewählt wird repräsentiert und eine Konstante (cursorRow, in Viergewinnt.asm deklariert, hat den Wert 6) für die vertikale-Position des Cursors direkt unterhalb des Spielfelds.

### 2.4.1 Aufbau

Der Cursor wird auf dem LCD durch einen 6 Byte Breiten, ausgefüllten Pfeil unter dem Spielfeld (Zeile 6) dargestellt (Abbildung 7).

	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte	8. Byte
1. Bit	1	1	1	1	1	1	1	1
2. Bit	2	2	2	2	2	2	2	2
3. Bit	3	3	3	3	3	3	3	3
4. Bit	4	4	4	4	4	4	4	4
5. Bit	5	5	5	5	5	5	5	5
6. Bit	6	6	6	6	6	6	6	6
7. Bit	7	7	7	7	7	7	7	7
8. Bit	8	8	8	8	8	8	8	8

Abbildung 2.6: CAPTION

### 2.4.2 Steuerung

Der Cursor startet nach betätigen des Resets (Taste 4 des Boards) oder bei Programmstart in der Mitte des Spielfeldes (Spalte 4). Er kann durch die Tasten 1 (nach links) und 3 (nach rechts) horizontal unter dem Spielfeld bewegt werden. Bei weiterer Bewegung und einer Cursorposition am Spielfeldrand erscheint der Cursor am gegenüberliegenden Spielfeldrand um schnelleres manövrieren

## 2 Programmiererhandbuch

zu ermöglichen (Abbildung 8). Bei Versetzen des Cursors wird zuerst auf dem LCD der alte Cursor gelöscht, dann die Variable `cursorColumn` für links um 1 reduziert oder für rechts um 1 erhöht. Danach wird der Cursor erneut auf dem LCD an der geänderten `cursorColumn` angezeigt.

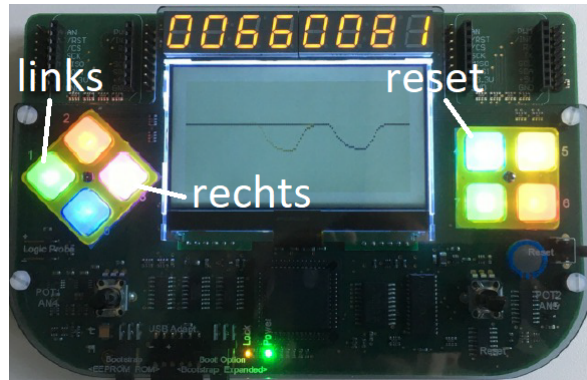


Abbildung 2.7: CAPTION

## 2.5 Logik

In dem folgenden Abschnitt wird die Spiellogik bzw. die genutzten Algorithmen im Detail erläutert. Darunter fallen Unterprogramme zur Adressberechnung, zum Prüfen ob eine Zelle leer ist, zum Spielerwechsel, zum Feststellen der Länge einer Steinfole sowie zum Feststellen des Spielendes.

### 2.5.1 Umrechnung der Boardadresse in eine Bufferadresse

Da das Spielfeld im Buffer Zeilenweise hintereinander weg verläuft, also alle 42 Zellen hintereinander liegen, aber auf dem Display die Zeilen untereinander angeordnet sind, müssen die Koordinaten umgerechnet werden können.

Zum umrechnen einer Spielfeldkoordinate wird folgende Formel verwendet:

$$Adresse = (Spalte \cdot 8) + (Zeile \cdot 56) \quad (2.6)$$

Die Multiplikatoren ergeben sich aus einer Zellenhöhe von 8 für die Zeilenzahl und aus einer Zellenbreite von 8 bei 7 Zellen pro Zeile = 56. Also bei einer Spielfeldkoordinate bei Spalte 3 und Zeile 2 wäre die Berechnung folgende:

$$136 = (3 \cdot 8) + (2 \cdot 56) \quad (2.7)$$

### 2.5.2 Prüfen des Zelleninhalts

Um Feststellen zu können ob ein Spieler eine zusammenhängende Steinfole besitzt, muss es nicht nur möglich sein zu erkennen ob eine Zelle leer oder belegt ist, sondern ebenfalls von welchem Spieler ein Stein ist.

Dazu wird erst im 3. Byte der Zelle (der Rand eines eventuellen Spielsteins für beide Varianten) geprüft ob nur Nullen (leere Zelle) oder auch Einsen (ein Spielstein von Spieler 1 oder Spieler 2) vorhanden sind.

Wenn ein Spielstein gefunden wurde, wird danach geprüft ob in der Mitte des Steins Einsen vorhanden sind (ausgefüllter Spielstein) oder nicht (innen leer) um ihn einem Spieler zuzuordnen.



Abbildung 2.8: Zellenbelegung

### 2.5.3 Spielerwechsel

Zum Wechsel des Spielers wird der aktuelle Spieler aus der Variable „player“ ausgelesen (Spieler 1 oder Spieler 2). Danach wird dann auf den jeweils anderen Spieler gewechselt.

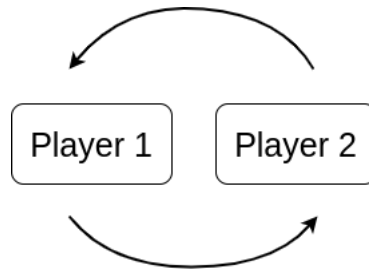


Abbildung 2.9: Spielerwechsel

### 2.5.4 Feststellen der Anzahl an zusammenhängenden Steinen

Da der Zustand eines gewonnenen Spiels nur direkt nach dem setzen eines Steins vorkommen kann, wird direkt nach jedem Zug darauf geprüft.

Um Festzustellen wie viele Steine zusammenhängend vom gleichen Spieler sind, werden immer von dem neu gelegten Stein zwei gegenüberliegende Seiten geprüft. Um die Gesamtzahl zu bestimmen muss am Ende noch eins abgezogen werden, damit der neu gelegte Stein nicht doppelt gezählt wird.

## 2 Programmiererhandbuch

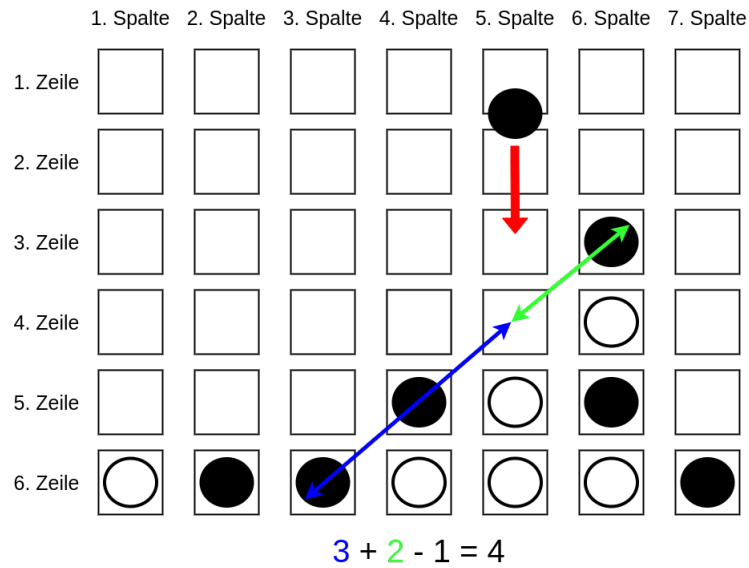


Abbildung 2.10: Zusammenhängend Steine

Eine Ausnahme bilden die Richtungen oben und unten. Da über dem neugelegten Stein keine weiteren Steine sein können, wird nur nach unten geprüft. Wenn bei dem Zählen der zusammenhängenden Steine mehr als vier erkannt werden, bedeutet dies das Spielende und es wird die Variable *playerWonFlag* auf 1 gesetzt um anschließend weitere Spielzüge zu sperren.

### 3 Listening

Listing 3.1: Viergewinnt.asm

```
1 ;-----
2 ; Viergewinnt.asm:
3 ;
4 ; @Author: Michael Persiehl (tinf102296), Guillaume Fournier-Mayer (tinf101922)
5 ; Implementiert das Spiel "4-Gewinnt" für 2 Spieler auf dem Motorola 68HC11.
6 ;-----
7
8
9 ;-----
10 ; Register- und Makrodefinitionen
11 ;-----
12
13 include trainer11Register.inc
14
15
16 ;-----
17 ; Definition von Variablen im RAM
18 ;-----
19
20 RamSection      Section
21                 org $0040
22
23 ramBegin        equ      *           ; Anfang des Rams
24 board           ds.b      boardSize  ; Der Buffer für das Spielfeld
25 cellAddress     ds.w      1           ; Die Adresse für eine Zeile des
26                                     ; Spielfeldes (als Funktionsparameter)
27
28 cursorColumn    ds.b      1           ; Horizontale Position des Cursors
29                                     ; in Pixeln
30
31 debugCellAdress ds.w      1           ; Die Adresse für eine Zeile des
32                                     ; Spielfeldes (Funktionsparameter
33                                     ; fürs Debugging)
34
35 buttonFlag      ds.b      1           ; Flag zur Flankenerkennung
36                                     ; -> 1 = gedrückt, 0 = nicht gedrückt
37
38 timer           ds.b      1           ; Timer zum entprellen der Tasten
39 player          ds.b      1           ; Der Spieler der aktuell am Zug ist
40                                     ; (1 = Spieler 1, 2 = Spieler 2)
41
42 coordx          ds.b      1           ; X-Koodrindate auf dem Spielfed
```

### 3 Listening

```

43 coordy          ds.b      1          ; Y-Koodrindate auf dem Spielfed
44 xoffset         ds.b      1          ; X-Offset für Linienerkennung
45 yoffset         ds.b      1          ; Y-Offset für Linienerkennung
46 playerWonFlag   ds.b      1          ; Flag um Tasten zu sperren wenn ein
47                                     ;Spieler gewonnen hat
48
49 RomSection      Section
50                 org $C000
51
52                 include Utils.inc
53                 include LCDUtils.inc
54                 include Board.inc
55                 include Cursor.inc
56                 include Input.inc
57                 include Logic.inc
58
59                 switch      RamSection
60 ramEnd          equ        *          ; Ende des Rams
61
62
63 ;-----
64 ;   Beginn des Programmcodes im schreibgeschuetzten Teil des Speichers
65 ;-----
66
67                 switch      RomSection
68 pageCmdMsk      dc.b        %10110000    ; Steuerkommando fürs Display
69 colCmdMskM      dc.b        %00010000    ; Steuerkommando fürs Display
70 colCmdMskL      dc.b        %00000000    ; Steuerkommando fürs Display
71 adrMask         dc.b        %00000111    ; Maske
72
73 LCDCols         equ        128           ; Horizontale Pixeln des LCD's
74 LCDRows         equ        8            ; Vertikalen Bytes des LCD's
75 boardSize       equ        336          ; Byteanzahl des Buffers
76 byteSize        equ        8           ; Größe eines Bytes in Bit
77 rowLength       equ        56           ; Zeilenlänge des Spielfeldes
78                                     ; in Pixeln
79
80 cursorRow       equ        6            ; Vertikale Position des Cursors
81                                     ; in Byte
82
83 boardOffset      equ        36           ; horizontaler Versatz des
84                                     ; Spielfeldes
85
86 boardMiddleColumn equ        24          ; Die mittlere Spalte des
87                                     ; Spielfeldes
88
89
90 ;-----
91 ; Konstanten für Textausgabe auf dem LCD
92 ; der erste Wert jeder Konstante repräsentiert deren Länge in Byte
93 ;-----

```

### 3 Listening

```

94
95 turnText
96     dc.b    21,$02,$7E,$02,$00    ; T
97     dc.b    $3C,$40,$40,$7C,$00  ; u
98     dc.b    $7C,$08,$04,$08,$00  ; r
99     dc.b    $7C,$08,$04,$78,$00  ; n
100    dc.b    $28,$00                ; :
101
102 playerText
103     dc.b    30,$7E,$12,$12,$0C,$00 ; P
104     dc.b    $02,$7E,$00            ; l
105     dc.b    $20,$54,$54,$78,$00    ; a
106     dc.b    $1C,$A0,$A0,$7C,$00    ; y
107     dc.b    $38,$54,$54,$48,$00    ; e
108     dc.b    $7C,$08,$04,$08,$00    ; r
109     dc.b    $00,$00                ;
110
111 oneText
112     dc.b    4,$04,$7E,$00,$00      ; 1
113
114 twoText
115     dc.b    5,$64,$52,$52,$4C,$00  ; 2
116
117 wonText
118     dc.b    17,$00,$00,$3C,$40,$20
119     , $40,$3C,$00                  ; w
120     dc.b    $38,$44,$44,$38,$00    ; o
121     dc.b    $7C,$08,$04,$78,$00    ; n
122
123
124 ; -----
125 ;   Hauptprogramm
126 ; -----
127
128 initGame
129
130     psha
131
132     ldaa    #$0C                    ; Löscht das Terminal
133     jsr     putChar
134     jsr     clrRam                  ; Überschreibt den Ram mit Nullen
135     jsr     initSPI                 ; Initialisierung der SPI-Schnittstelle
136     jsr     initLCD                 ; Initialisierung des Displays
137     jsr     LCDClr                  ; Löscht das Display
138     jsr     createGrid              ; Schreibt das Spielfeld in den Buffer
139     jsr     drawBoard               ; Schreibt den Bufferinhalt auf den LCDs
140     jsr     resetCursor             ; Setz den Cursor in die Mitte des
141                                     ; Spielfeldes
142     ldaa    #1
143     staa    player                  ; wechselt zu Spieler 1
144     jsr     showText

```



### 3 Listening

```
145         jsr     drawPlayer
146         pula
147
148         rts
149
150 Start
151         lds     $$3FFF                ; Einstiegspunkt des Spieles
152         jsr     initGame
153
154 mainLoop
155         jsr     readInput
156         bra     mainLoop
157
158 End     bra     *
159
160
161 ;-----
162 ;   Vektortabelle
163 ;-----
164 VSection      Section
165         org     VRESET                ; Reset-Vektor
166         dc.w     Start                ; Programm-Startadresse
```

### 3 Listening

Listing 3.2: Board.inc

```
1      switch RomSection
2  ; -----
3  ; Board.inc:
4  ;
5  ; Verwaltet den Buffer, welcher das Spielfeld repräsentiert.
6  ; Bietet Funktionen um den Buffer zu befüllen und diesen auf dem LCD zu
7  ; Zeichnen.
8  ; -----
9
10
11 ; -----
12 ; drawCell:
13 ;
14 ; Zeichnet eine Zelle auf den LCD-Display.
15 ;
16 ; @param cellAddress: Relative Zellenadresse vom Anfang des Buffers
17 ; -----
18
19 drawCell
20     pshx
21     pshy
22     psha
23     pshb
24
25     ; Berechne die absolute Adresse der Zelle im Buffer
26     ldy    #board
27     xgdy
28     addd   cellAddress
29     xgdy
30
31     ; Zeile und Spalte der Zelle berechnen und in D (Spalte)
32     ; und X (Zeile) Speichern
33     ldd    cellAddress
34     ldx    #rowLength
35     idiv
36
37
38     psha
39     pshb
40
41     ; Zeile setzen
42     xgdx
43     tba
44     jsr    LCDSetRow
45
46     pulb
47     pula
48
49     ; Bildschirm Versatz mit berücksichtigen
50     tba
```

### 3 Listening

```
51         adda    #boardOffset
52         ldb     #0
53 drawCellLoop
54
55         ; Spalte setzen
56         jsr     LCDSetCol
57
58         ; Zeichen (Byte) an LCD senden
59         psha
60         lda     0,y
61         jsr     LCDdata
62         pula
63
64         iny
65         inca
66         incb
67         cmpb    #8
68         bne     drawCellLoop
69
70         pulb
71         pula
72         puly
73         pulx
74         rts
75
76
77 ; -----
78 ; cellToBuffer:
79 ;
80 ; Schreibt eine leere "Zelle" (8 * 8 Bit) an der Position angegeben
81 ; durch "cellAddress" in den Buffer "Board".
82 ;
83 ; @param cellAddress: Relative Zellenadresse vom Anfang des Buffers
84 ; -----
85
86 cellToBuffer
87         pshx
88         psha
89         pshb
90
91         ; Berechne Adresse der Zelle im Buffer
92         ldx     #board
93         xgdx
94         addd    cellAddress
95         xgdx
96
97
98         lda     #%11111111      ; linker Rand der Zelle
99         staa    0,x
100
101         lda     #%10000001
```

### 3 Listening

```
102      staa    1,x
103
104      lda     #%10000001
105      staa    2,x
106
107      lda     #%10000001
108      staa    3,x
109
110      lda     #%10000001
111      staa    4,x
112
113      lda     #%10000001
114      staa    5,x
115
116      lda     #%10000001
117      staa    6,x
118
119      lda     #%11111111      ; rechter Rand der Zelle
120      staa    7,x
121
122      pulb
123      pula
124      pulx
125      rts
126
127
128      ; -----
129      ; creatGrid:
130      ;
131      ; Füllt den komplettenden Buffer "Board" mit leeren Zellen.
132      ; -----
133
134      createGrid
135          psha
136          pshb
137
138          ldd     #0
139          std     cellAddress
140      createGridLoop
141          jsr     cellToBuffer
142          addd     #byteSize
143          std     cellAddress
144          cmpd     #boardSize
145          bne     createGridLoop
146
147          pula
148          pulb
149          rts
150
151
152      ; -----
```

### 3 Listening

```
153 ; drawBoard:
154 ;
155 ; Zeichnet den kompletten Buffer auf den Display.
156 ; -----
157
158 drawBoard
159     psha
160     pshb
161
162     ldd    #0
163     std    cellAddress
164 drawBoardLoop
165     jsr    drawCell
166     addd   #byteSize
167     std    cellAddress
168     cmpd   #boardSize
169     bne    drawBoardLoop
170
171     pula
172     pulb
173     rts
174
175 ; -----
176 ; stone1ToBuffer:
177 ;
178 ; Schreibt einen Spielstein für Spieler1 (gefüllt) in den Buffer,
179 ; der Spielstein wird dabei mit der leeren Zelle oder-Verknüpft.
180 ;
181 ; @param cellAddress: Relative Zellenadresse vom Anfang des Buffers
182 ; -----
183
184 stone1ToBuffer
185     pshx
186     psha
187     pshb
188
189     ; Berechne Adresse der Zelle im Buffer
190     ldx    #board
191     xgdx
192     addd   cellAddress
193     xgdx
194
195     lda    #%00011000    ; Stein für Spieler 1 (gefüllt)
196     oraa   2,x
197     staa   2,x
198
199     lda    #%00111100
200     oraa   3,x
201     staa   3,x
202
203     lda    #%00111100
```

### 3 Listening

```
204      oraa      4,x
205      staa      4,x
206
207      lda       #%00011000
208      oraa      5,x
209      staa      5,x
210
211      pulb
212      pula
213      pulx
214      rts
215
216
217 ; -----
218 ; stone2ToBuffer:
219 ;
220 ; Schreibt einen Spielstein für Spieler2 (leer) in den Buffer,
221 ; der Spielstein wird dabei mit der leeren Zelle oder-Verknüpft.
222 ;
223 ; @param cellAddress: Relative Zellenadresse vom Anfang des Buffers
224 ; -----
225
226 stone2ToBuffer
227     pshx
228     psha
229     pshb
230
231     ; Berechne Adresse der Zelle im Buffer
232     ldx        #board
233     xgdx
234     addd       cellAddress
235     xgdx
236
237     lda        #%00011000      ; Stein für Spieler 2 (leer)
238     oraa       2,x
239     staa       2,x
240
241     lda        #%00100100
242     oraa       3,x
243     staa       3,x
244
245     lda        #%00100100
246     oraa       4,x
247     staa       4,x
248
249
250     lda        #%00011000
251     oraa       5,x
252     staa       5,x
253
254     pulb
```

### 3 *Listening*

```
255      pula
256      pulx
257      rts
```

### 3 Listening

Listing 3.3: Debug.inc

```
1      switch RomSection
2  ; -----
3  ; Debug.inc:
4  ; Bietet Funktionen zum Debugging. Gibt Akkumulatoren, Register und CCR aus.
5  ; Basisfunktionen von J.Voelker.
6  ; Ergänzt um debugBuffer und debugCell.
7  ; -----
8
9
10 ; -----
11 ;   Vier Bits - ein Nibble - aus B als Hexadezimalzahl ausgeben
12 ; -----
13
14 nibbletohex      dc.b      "0123456789ABCDEF"
15
16 putNibble        pshy
17                  psha
18                  pshb
19
20                  andb #$0F          ; Nibble maskieren,
21                  ldy #nibbletohex   ; aus der Zeichentabelle
22                  aby                 ; das passende Zeichen waehlen
23                  ldaa 0,y            ; und in A laden.
24
25                  jsr putChar
26
27                  pulb
28                  pula
29                  puly
30                  rts
31
32 ; -----
33 ;   Inhalt von B als Hexadezimalzahl ausgeben
34 ; -----
35
36 putByte          pshb
37
38                  rorb                ; Oberes Nibble von B zur Ausgabe vorbereiten
39                  rorb
40                  rorb
41                  rorb
42                  jsr putNibble
43
44                  pulb
45
46                  jsr putNibble        ; Unteres Nibble von B ausgeben
47                  rts
48
49 ; -----
50 ;   Inhalt von D als Hexadezimalzahl ausgeben
```



### 3 Listening

```
51 ; -----
52
53 putHex      psha
54             ldaa #'$'
55             jsr putChar
56             pula
57
58             pshb
59             tab                ; Zuerst A ausgeben, den oberen Teil von D
60             jsr putByte
61             pulb
62
63             jsr putByte        ; Anschliessend B ausgeben, den unteren
64                                 ; Teil von D
65
66             jsr space
67             rts
68
69 ; -----
70 ;   Inhalt von D als Hexadezimalzahl ausgeben
71 ; -----
72
73 putHexBuffer
74             pshb
75             tab                ; Zuerst A ausgeben, den oberen Teil von D
76             jsr putByte
77             pulb
78
79             jsr putByte        ; Anschliessend B ausgeben, den unteren
80                                 ; Teil von D
81
82             jsr space
83             rts
84
85 ; -----
86 ;   Carry-Flag als Binaerzahl ausgeben
87 ; -----
88
89
90 putCarry     psha
91
92             bcs putC1          ; War das Carry-Flag gesetzt ?
93
94             ldaa #'0'
95             bra putCAusgabe
96
97 putC1        ldaa #'1'
98
99 putCAusgabe  jsr putChar
100
101             pula
```

### 3 Listening

```
102         rts
103
104 ; -----
105 ;   Wert in A als Binaerzahl ausgeben
106 ; -----
107
108 putBinary    psha
109              ldaa #'%'
110              jsr putChar
111              pula
112
113              psha
114
115              rola                ; Das oberste Bit ins Carry-Flag rotieren
116              bsr putCarry        ; und als Binaerzahl ausgeben.
117              rola
118              bsr putCarry
119              rola
120              bsr putCarry
121              rola
122              bsr putCarry
123              rola
124              bsr putCarry
125              rola
126              bsr putCarry
127              rola
128              bsr putCarry
129              rola
130              bsr putCarry
131
132              ldaa #32            ; Leerzeichen anfüegen
133              jsr putChar
134
135              pula
136              rts
137
138 ; -----
139 ;   Alle Register ausgeben, dabei nichts veraendern, auch die Flags
140 ;   bleiben erhalten
141 ; -----
142
143 debug        ; Alter PC          (Stackpointer + 7)
144              pshy                ; (Stackpointer + 5)
145              pshx                ; (Stackpointer + 3)
146              pshb                ; (Stackpointer + 2)
147              psha                ; (Stackpointer + 1)
148              tpa                  ;
149              psha                ; (Stackpointer + 0)   Flags in A holen
150                                   und ebenfalls sichern.
151 ; -----
152
```

### 3 Listening

```
153      tsx                ; Stackpointer in X holen
154
155      ldaa #'P'          ; PC=
156      jsr putChar
157      ldaa #'C'
158      jsr putChar
159      ldaa #'='
160      jsr putChar
161
162      ldd 7,x            ; Den alten PC vom Stack in D laden
163      jsr putHex
164
165      ; -----
166
167      ldaa #'D'          ; D=
168      jsr putChar
169      ldaa #'='
170      jsr putChar
171
172      ldd 1,x            ; Gesicherten Inhalt von D vom Stack laden
173      jsr putHex
174
175      ; -----
176
177      ldaa #'X'          ; X=
178      jsr putChar
179      ldaa #'='
180      jsr putChar
181
182      ldd 3,x            ; Gesicherten Inhalt von X vom Stack laden
183      jsr putHex
184
185      ; -----
186
187      ldaa #'Y'          ; Y=
188      jsr putChar
189      ldaa #'='
190      jsr putChar
191
192      ldd 5,x            ; Gesicherten Inhalt von Y vom Stack laden
193      jsr putHex
194
195      ; -----
196
197      ldaa #'S'          ; SP=
198      jsr putChar
199      ldaa #'P'
200      jsr putChar
201      ldaa #'='
202      jsr putChar
203
```

### 3 Listening

```
204      xgdx                ; Den Stackpointer selbst in D laden
205      jsr putHex
206
207      ; -----
208
209      ldaa #'C'           ; CCR=
210      jsr putChar
211      ldaa #'C'
212      jsr putChar
213      ldaa #'R'
214      jsr putChar
215      ldaa #'='
216      jsr putChar
217
218      tsx                ; Nochmal den Stackpointer in X holen
219      ldaa 0,x            ; Die auf dem Stack gesicherten Flags
220                        ; in A laden
221
222      jsr putBinary
223
224      ; -----
225
226      jsr crlf
227
228      pula               ; Alten Inhalt
229      tap                ; der Flags zurueckholen
230      pula
231      pulb
232      pulx
233      puly               ; Alle Register wiederhergestellt.
234      rts                ; Nach dem Ruecksprung ist auch der
235                        ; Stackpointer wieder wie vorher.
236
237
238 ; -----
239 ; debugBuffer:
240 ;
241 ; Gibt den gesamten Buffer in Hexadezimal auf dem Terminal aus.
242 ;
243 ; @param board: Das Board (Buffer)
244 ; -----
245 debugBuffer
246     pshx
247     psha
248     pshb
249
250     ldx    #board
251     ldd    #336
252 debugBufferLoop
253     psha
254     pshb
```

### 3 Listening

```
255         ldd      0,x
256         jsr      putHexBuffer
257         pulb
258         pula
259         inx
260         inx
261         subd     #2
262         bne      debugBufferLoop
263
264         pulb
265         pula
266         pulx
267         rts
268
269
270
271 ; -----
272 ; debugCell:
273 ;
274 ; Gibt eine Zelle des Buffers in Hexadezimal auf dem Terminal aus.
275 ;
276 ; @param debugCellAdress: Relative Zellenadresse vom Anfang des Buffers
277 ; @param board: Das Board (Buffer)
278 ; -----
279 debugCell
280     pshx
281     psha
282     pshb
283
284     ldx      #board
285     xgdx
286     addd     debugCellAdress
287     xgdx
288
289     ldd      #8
290 debugCellLoop
291     psha
292     pshb
293     ldd      0,x
294     jsr      putHexBuffer
295     pulb
296     pula
297     inx
298     inx
299     subd     #2
300     bne      debugCellLoop
301
302     jsr      crlf
303     pulb
304     pula
305     pulx
```



### 3 Listening

Listing 3.4: Input.inc

```

1      switch RomSection
2      ; -----
3      ; Input.inc:
4      ;
5      ; Verarbeitet die Benutzereingaben und reagiert dementsprechend.
6      ; -----
7
8
9      ; -----
10     ; readInput:
11     ;
12     ; Liest das Datenbyte des Tastenfeldes (PIO_B) aus und reagiert entsprechend
13     ; auf die Taste.
14     ;
15     ; @param PIO_B: Das Signal Byte der 8 Tasten in 1 aus n Kodierung
16     ; -----
17
18 readInput
19     psha
20     pshb
21
22     ldaa PIO_B           ; Lese Button byte ein
23     ldab buttonFlag      ; Lese ButtonFlag ein
24
25     cmpb #1              ; 1 Wenn Taste gedrückt
26     beq testButtonRelease
27
28 button4                  ; Wenn Taste 4 gedrückt, reset
29                          ; des Spiels
30     cmpa #%11101111
31     bne button0
32     jsr setButtonFlag
33     jsr initGame
34     bra readInputEnd
35
36 button0                  ; Wenn Taste 0 gedrückt, setze Stein
37     tst playerWonFlag    ; alle Tasten ausser reset sperren wenn
38                          ; das Spiel gewonnen ist
39     bne readInputEnd
40
41     cmpa #%11111110
42     bne button1
43     jsr setButtonFlag
44     jsr setStone
45
46     ldab coordy          ; testen ob ein Stein gesetzt werden
47                          ; konnte, wenn nicht KEIN
48                          ; Spielerwechsel / checkWin
49     cmpb #$FF
50     beq readInputEnd

```

### 3 Listening

```

51
52     jsr     checkWin
53     cmpa    #1
54     beq     someoneWon
55
56     jsr     nextPlayer
57     jsr     drawPlayer
58
59     bra     readInputEnd
60
61 button1                                     ; Wenn Taste 1 gedrückt, setze Flag
62                                     ; und bewege Cursor nach Links
63     cmpa    #%11111101
64     bne     button3
65     jsr     setButtonFlag
66     jsr     moveCursorLeft
67     bra     readInputEnd
68 button3                                     ; Wenn Taste 3 gedrückt, setze Flag
69                                     ; und bewege Cursor nach Rechts
70     cmpa    #%11110111
71     bne     readInputEnd
72     jsr     setButtonFlag
73     jsr     moveCursorRight
74     bra     readInputEnd
75
76
77 testButtonRelease
78     jsr     incTimer                       ; entprellen der Tasten mit einem Timer
79     ldab    timer
80     cmpb    #250
81     bne     readInputEnd
82     jsr     resetTimer
83
84     cmpa    #%11111111                     ; Überprüfe ob Taste losgelassen
85                                     ; wurde -> Reset Buttonflag
86     bne     readInputEnd
87     jsr     resetButtonFlag
88
89
90
91     bra     readInputEnd
92 someoneWon
93     jsr     playerwon
94
95 readInputEnd
96
97     pulb
98     pula
99     rts
100
101

```



### 3 Listening

```
102 ; -----
103 ; setButtonFlag:
104 ;
105 ; Setzt ein Flag das eine Taste gedrückt -und bisher noch nicht
106 ; losgelassen wurde.
107 ;
108 ; @return buttonFlag: auf 1 gesetzt
109 ; -----
110
111 setButtonFlag
112     psha
113
114     ldaa    #1
115     staa    buttonFlag
116
117     pula
118     rts
119
120
121 ; -----
122 ; resetButtonFlag:
123 ;
124 ; Setzt das Buttonflag zurück.
125 ;
126 ; @return buttonFlag: auf 0 gesetzt
127 ; -----
128
129 resetButtonFlag
130     psha
131
132     ldaa    #0
133     staa    buttonFlag
134
135     pula
136     rts
137
138
139 ; -----
140 ; incTimer:
141 ;
142 ; Inkrementiert einen Timer zum entprellen der Tasten.
143 ;
144 ; @param timer: der Timer der inkrementiert werden soll
145 ; @return timer: der inkrementierte Timer
146 ; -----
147
148 incTimer
149     psha
150
151     ldaa    timer
152     adda    #1
```

### 3 Listening

```
153         staa     timer
154
155     pula
156     rts
157
158
159 ; -----
160 ; resetTimer:
161 ;
162 ; Setzt den Timer zum entprellen der Tasten zurück.
163 ;
164 ; @param timer: der Timer der zurückgesetzt werden soll
165 ; @return timer: der zurückgesetzte Timer
166 ; -----
167
168 resetTimer
169     psha
170
171     ldaa     #0
172     staa     timer
173
174     pula
175     rts
```

### 3 Listening

Listing 3.5: LCDutils.inc

```
1      switch RomSection
2  ; -----
3  ; LCDutils.inc:
4  ;
5  ; Bietet Funktionen um das Display zu initialisieren und anzusteuern.
6  ; -----
7
8
9  ; -----
10 ; initSPI:
11 ;
12 ; Initialisiert die SPI-Schnittstelle.
13 ; -----
14
15 initSPI
16     psha
17     ldaa #%00000011
18     staa PORTD
19     ldaa #%00111010
20     staa DDRD ; TXD,MOSI,SCK,-SS = out
21     ldaa #%01010010 ; 125kHz,Mode=0
22     staa SPCR ; enable,no int
23     pula
24     rts
25
26
27 ; -----
28 ; SPItransfer:
29 ;
30 ; Hilfsmethode für LCDsend
31 ; -----
32
33 SPItransfer
34     staa SPDR ; Byte senden
35
36 SPIwait2
37     tst SPSR ; Warten mal anders
38     bpl SPIwait2 ; auf MSB = SPIF = 1
39     ldaa SPDR ; Antwortbyte holen
40     rts
41
42
43 ; -----
44 ; initLCD:
45 ;
46 ; Initialisierung des Displays.
47 ; -----
48
49 initLCD
50     psha
51     ldaa #$A2
```

### 3 Listening

```
51         bsr LCDcommand
52         ldaa #$A0
53         bsr LCDcommand
54         ldaa #$C8
55         bsr LCDcommand
56         ldaa #$24
57         bsr LCDcommand
58         ldaa #$81
59         bsr LCDcommand
60         ldaa #$2F
61         bsr LCDcommand
62         ldaa #$2F
63         bsr LCDcommand
64         ldaa #$AF
65         bsr LCDcommand
66         bclr PORTA,%01000000      ; PA6 = DIMM
67         pula                      ; Hintergrundbeleuchtung an
68         rts
69
70
71 ; -----
72 ; LCDcommand:
73 ;
74 ; Sendet ein Kommandobyte an das Display.
75 ;
76 ; @param Akku a: Das zu sendende Kommandobyte
77 ; -----
78
79 LCDcommand
80         bclr PORTD,%00100000 ; PD5, LCD_A0=0
81         bra LCDsend
82
83
84 ; -----
85 ; LCDdata:
86 ;
87 ; Sendet ein Datenbyte an das Display.
88 ;
89 ; @param Akku a: Das zu sendende Datenbyte
90 ; -----
91
92 LCDdata
93         bset PORTD,%00100000 ; PD5, LCD_A0=1
94         bra LCDsend
95
96
97 ; -----
98 ; LCDSend:
99 ;
100 ; Hilfsmethode für LCDcommand und LCDdata.
101 ; Sendet ein Daten- oder Kommando an das Display.
```

### 3 Listening

```
102 ;
103 ; @param Akku a: Das zu sendende Datenbyte
104 ; -----
105
106 LCDSend
107     psha
108     ldaa    PIO_C
109     anda    #%00111111 ; SPI_SEL = 0 = LCD
110     staa    PIO_C
111     pula
112
113     psha
114     jsr     SPItransfer
115
116     ldaa    PIO_C ; Deselect LCD
117     oraa    #%11000000 ; SPI_SEL = 3 = EEPROM
118     staa    PIO_C
119
120     pula
121     rts
122
123 ; -----
124 ; LCDSetRow:
125 ;
126 ; Setzt die Zeile des LCD-Cursors.
127 ;
128 ; @param Akku a: Die Cursorzeile
129 ; -----
130 LCDSetRow
131     psha
132     oraa    pageCmdMsk
133     jsr     LCDcommand
134     pula
135     rts
136
137
138 ; -----
139 ; LCDsetColMSB:
140 ;
141 ; Hilfsmethode für LCDSetCol.
142 ; Setzt die ersten vier Bits der Spalte des LCD-Cursors.
143 ;
144 ; @param Akku a: die ersten vier Bits der Spalte des LCD-Cursors
145 ; -----
146
147 LCDsetColMSB
148     psha
149     oraa    colCmdMskM
150     jsr     LCDcommand
151     pula
152     rts
```

### 3 Listening

```
153
154
155 ; -----
156 ; LCDsetColLSB:
157 ;
158 ; Hilfsmethode für LCDSetCol.
159 ; Setzt die letzten vier Bits der Spalte des LCD-Cursors.
160 ;
161 ; @param Akku a: die letzten vier Bits der Spalte des LCD-Cursors
162 ; -----
163
164 LCDsetColLSB
165     psha
166     oraa    colCmdMskL
167     jsr     LCDcommand
168     pula
169     rts
170
171
172 ; -----
173 ; LCDSetCol:
174 ;
175 ; Setzt die Spalte des LCD-Cursors.
176 ;
177 ; @param Akku a: die Spalte des LCD-Cursors
178 ; -----
179
180 LCDSetCol
181     psha
182     psha
183     anda    #%00001111
184     jsr     LCDsetColLSB
185     pula
186     lsra    ;Shifte um 4 nach links
187     lsra
188     lsra
189     lsra
190     jsr     LCDSetColMSB
191     pula
192     rts
193
194
195 ; -----
196 ; LCDClrRow:
197 ;
198 ; Hilfsmethode für LCDClr.
199 ; -----
200
201 LCDClrRow
202     psha
203     pshb
```

### 3 Listening

```
204         ldab    #LCDCols
205         ldaa    #0
206         jsr     LCDSetCol
207
208 LCDClrRowFor
209         jsr     LCDdata
210         decb
211         bne     LCDClrRowFor
212
213         pulb
214         pula
215         rts
216
217
218 ; -----
219 ; LCDClr:
220 ;
221 ; Löscht alle Pixel des LCD's.
222 ; -----
223
224 LCDClr
225         psha
226         ldaa    #0
227 LCDClrFor
228         jsr     LCDSetRow
229         jsr     LCDClrRow
230         inca
231         cmpa    #LCDRows
232         bne     LCDClrFor
233
234         ldaa    #0           ; Setze Cursor wieder auf 0:0
235         jsr     LCDSetCol
236         jsr     LCDSetRow
237         pula
238         rts
239
240
241 ; -----
242 ; showText:
243 ;
244 ; Schreibt die Texte für "Player" und "Turn:" auf den LCD.
245 ; -----
246
247 showText
248
249         psha
250         pshb
251         pshx
252
253
254         ldd     #0
```

### 3 Listening

```
255         ldx    #turnText      ; Text "Turn:" oben links auf LCD
256         jsr    drawText
257
258
259         ldaa    #1
260         ldab    #0
261         ldx    #playerText    ; Text "Player" in die 2. Zeile, links auf LCD
262         jsr    drawText
263
264         pulx
265         pulb
266         pula
267
268         rts
269
270
271 ; -----
272 ; drawText:
273 ;
274 ; Zeichnet den Text an die übergebene Zeile und Spalte des Displays
275 ;
276 ; @param Akku a: Zeile des LCD's
277 ; @param Akku b: Spalte des LCD's
278 ; @param Register x: Adresse des Textes im Speicher
279 ; -----
280
281 drawText
282
283         psha
284         pshb
285         pshx
286
287         jsr    LCDSetRow
288         tba
289         ldab    0,x
290         inx
291         ; nach setzen der Zeile, Spalte in A schreiben
292         ; erstes Byte in X ist die Länge des Texts
293
294 drawTextLoop
295
296         jsr    LCDSetCol
297         psha
298         ldaa    0,x
299         jsr    LCDdata
300         pula
301         inca
302         inx
303         decb
304         bne    drawTextLoop
305
306         pulx
307         pulb
```



### 3 Listening

```
306         pula
307         rts
308
309
310 ; -----
311 ; drawPlayer:
312 ;
313 ; Schreibt die Spielernummer des Spielers, welcher gerade am Zug ist.
314 ;
315 ; @param player: Die Nummer des Spielers, welcher gerade am Zug ist
316 ; -----
317
318 drawPlayer
319
320         psha
321         pshb
322         pshx
323         pshy
324
325
326         ldaa    #1
327         ldy     #playerText
328         ldab    0,y
329         incb
330
331         psha
332         ldaa    player
333         cmpa    #2
334         pula
335
336         beq     drawPlayer2
337
338 drawPlayer1
339         ldx     #oneText
340         jsr     drawText
341         bra     drawPlayerEnd
342 drawPlayer2
343         ldx     #twoText
344         jsr     drawText
345 drawPlayerEnd
346
347         puly
348         pulx
349         pulb
350         pula
351
352         rts
353
354
355 ; -----
356 ; drawWinner:
```

### 3 Listening

```
357 ;
358 ; Schreibt eine Nachricht an den Spieler, welcher das Spiel gewonnen hat,
359 ; auf den LCD.
360 ; Löscht alle anderen Texte vom LCD und sperrt alle Tasten außer der
361 ; Reset-Taste.
362 ;
363 ; @param player: Die Nummer des Spielers, welcher gewonnen hat
364 ; -----
365
366 drawWinner
367     psha
368     pshb
369     pshx
370
371     ldaa    #6                ; Zeichne "Spieler"
372     ldab    #boardOffset
373     ldx     #playerText
374     jsr     drawText
375
376
377     addb    0,x                ; Zeichne die Spielernummer
378     psha
379     ldaa    player
380     cmpa    #2
381     pula
382     beq     drawWinner2      ; bestimmen welcher Spieler gewonnen hat
383
384
385 drawWinner1                    ; Fall Spieler1 gewonnen
386     ldx     #oneText
387     jsr     drawText
388     bra     drawNumberEnd
389
390 drawWinner2                    ; Fall Spieler2 gewonnen
391     ldx     #twoText
392     jsr     drawText
393
394
395 drawNumberEnd
396     addb    0,x
397     ldx     #wonText
398     jsr     drawText
399
400     pulx
401     pulb
402     pula
403     rts
```

### 3 Listening

Listing 3.6: Logic.inc

```
1      switch RomSection
2  ; -----
3  ; Logic.inc:
4  ;
5  ; Bietet alle Funktionen zur Spiellogik (Berechnen des Gewinners,
6  ; Zugwechsel, etc.).
7  ; -----
8
9
10 ; -----
11 ; getCellAddress:
12 ;
13 ; Rechnet eine Spielfeldkoordinate (X und y) in eine Adresse im Buffer um.
14 ;
15 ; @param Akku a: X-Wert der Spielfeldkoordinate
16 ; @param Akku b: Y-Wert der Spielfeldkoordinate
17 ; @return cellAddress: Die Adresse der Zelle relativ zur Bufferadresse
18 ; -----
19
20 getCellAddress                ; Address = (b * 56) + (a * byteSize)
21     psha
22     pshb
23     pshb
24
25     ldab    #8
26     mul
27     std     cellAddress
28
29     pulb
30
31     ldaa    #rowLength
32     mul
33     addd    cellAddress
34     std     cellAddress
35
36     pulb
37     pula
38     rts
39
40
41 ; -----
42 ; setStone:
43 ;
44 ; Prüft, ob in die Spalte an der aktuellen Cursorposition ein Stein
45 ; gelegt werden kann. Wenn die Position gültig ist, wird ein Stein
46 ; gesetzt.
47 ;
48 ; @param cursorColumn: Die Spalte, in die ein Stein gesetzt werden soll
49 ; -----
50
```

### 3 Listening

```
51 setStone
52     pshb
53     psha
54     pshx
55     pshy
56
57
58     ldaa    #0
59     ldab    cursorColumn
60     subb    #boardOffset
61
62     ldx     #byteSize
63     idiv
64     xgdx
65     tba
66     ldab    #5
67
68     ldy     #6
69 freeCellLoop
70     jsr     getCellAddress    ; Ergebnis in Variable cellAddress
71
72     psha
73     jsr     getCellOccupancy
74     cmpa    #0
75
76     pula
77     bne     checkNextCell
78
79     psha
80     ldaa    player           ; Überprüfe welcher Spieler dran ist
81     cmpa    #1
82     pula
83     bne     Player2Stone
84
85 Player1Stone
86     jsr     stone1ToBuffer
87     bra     setStoneEnd
88
89 Player2Stone
90     jsr     stone2ToBuffer
91     bra     setStoneEnd
92
93 checkNextCell
94
95     decb
96     dey
97     bne     freeCellLoop
98
99 setStoneEnd
100
101     ; speichern der Spielfeldkoordinaten zum prüfen ob gewonnen wurde
```

### 3 Listening

```
102      staa    coordx
103      stab    coordy
104
105      jsr     drawBoard
106
107      puly
108      pulx
109      pula
110      pulb
111      rts
112
113
114 ; -----
115 ; getCellOccupancy:
116 ;
117 ; Gibt die Belegung einer Zeller wieder.
118 ;
119 ; @param cellAddress: Die Zellenadresse
120 ; @return Akku a: 0 = leer, 1 = Spieler1, 2 = Spieler2
121 ; -----
122
123 getCellOccupancy                                ; Rückgabewert in a
124     pshb
125     pshx
126
127     ldx     #board
128     xgdx
129     addd    cellAddress
130     xgdx
131     ; 3. Byte der Zelle: prüfen ob Zelle nur Nullen enthält (keinen Stein)
132     lda     3,x
133     ; erste und letzte 1 repräsentieren den Zellenrand
134     cmpa    #%10000001
135     beq     isEmpty
136     cmpa    #%10111101
137     beq     isPlayer1
138     cmpa    #%10100101
139     beq     isPlayer2
140
141 isEmpty
142     ldaa    #0
143     bra     getCellOccupancyEnd
144
145 isPlayer1
146     ldaa    #1
147     bra     getCellOccupancyEnd
148
149 isPlayer2
150     ldaa    #2
151     bra     getCellOccupancyEnd
152
```

### 3 Listening

```
153 getCellOccupancyEnd
154
155      pulx
156      pulb
157      rts
158
159
160
161  ; -----
162  ; nextPlayer:
163  ;
164  ; Wechselt den aktuellen Spieler. Wenn Spieler1 am Zug ist, wird auf
165  ; Spieler2 gewechselt und umgekehrt.
166  ;
167  ; @param player: Der Spieler, der aktuell am Zug ist
168  ; @return player: 1 = Spieler1, 2 = Spieler2
169  ; -----
170
171  nextPlayer
172      psha
173
174      ldaa    player
175      cmpa    #1
176      beq     setPlayer2
177
178  setPlayer1
179      ldaa    #1
180      staa    player
181      bra     nextPlayerEnd
182
183  setPlayer2
184      ldaa    #2
185      staa    player
186
187  nextPlayerEnd
188
189      pula
190      rts
191
192
193  ; -----
194  ; checkWin:
195  ;
196  ; Überprüft, ob der aktuelle Spieler eine Viererreihe erreicht hat
197  ;
198  ; @param coordx: Die X-Koordinate des gesetzten Steins
199  ; @param coordy: Die Y-Koordinate des gesetzten Steins
200  ; @return Akku a: 0 = False, 1 = True
201  ; -----
202
203  checkWin
```

### 3 Listening

```
204
205     pshb
206
207
208     ; Vertikal
209     ldaa    #0                ; Richtung unten
210     ldab    #1
211     jsr     getLineLength
212     cmpa    #4
213     bhs     checkWinTrue
214
215
216     ; Horizontal
217     ldaa    #1                ; Richtung Rechts
218     ldab    #0
219     jsr     getLineLength
220     tab
221     pshb
222     ldaa    #-1               ; Richtung Links
223     ldab    #0
224     jsr     getLineLength
225     pulb
226     deca
227     aba
228     cmpa    #4
229     bhs     checkWinTrue
230
231
232     ; Diagonal Oben Rechts nach unten Links
233     ldaa    #1                ; Richtung Oben Rechts
234     ldab    #-1
235     jsr     getLineLength
236     tab
237     pshb
238     ldaa    #-1               ; Richtung unten Links
239     ldab    #1
240     jsr     getLineLength
241     pulb
242     deca
243     aba
244     cmpa    #4
245     bhs     checkWinTrue
246
247
248     ; Diagonal unten Rechts nach links Oben
249     ldaa    #-1               ; Richtung links Oben
250     ldab    #-1
251     jsr     getLineLength
252     tab
253     pshb
254     ldaa    #1                ; Richtung unten Rechts
```

### 3 Listening

```
255         ldab    #1
256         jsr     getLineLength
257         pulb
258         deca
259         aba
260         cmpa    #4
261         bhs     checkWinTrue
262
263         bra     checkWinFalse
264
265 checkWinTrue
266         ldaa    #1
267         staa    playerWonFlag
268         bra     checkWinEnd
269
270 checkWinFalse
271         ldaa    #0
272
273 checkWinEnd
274
275         pulb
276         rts
277
278
279 ; -----
280 ; getLineLength:
281 ;
282 ; Zählt die Anzahl der verbundenen Steine des aktuellen Spielers in einer
283 ; Reihe.
284 ;
285 ; @param Akku a: X-Richtung in die gelaufen werden soll
286 ; @param Akku b: Y-Richtung in die gelaufen werden soll
287 ; @param player: Der Spieler, der aktuell am Zug ist
288 ; @return Akku a: Anzahl an Steinen die in einer Reihe gefunden wurden
289 ; -----
290
291 getLineLength
292
293         pshb
294         pshy
295         pshx
296
297         staa    xoffset
298         stab    yoffset
299         ldaa    coordx
300         ldab    coordy
301
302         ldx     #0                                ; Zähler für die Länge einer Linie
303
304 getLineLengthLoop
305
```



### 3 Listening

```
306      adda    xoffset
307      addb    yoffset
308
309      psha
310      jsr      isOutOfBorder      ; Überprüfe ob nächster Stein sich noch
311      cmpa     #1                  ; innerhalb des Spielfeldes befindet
312      pula
313      beq      getLineLengthLoopEnd
314
315      jsr      getCellAddress
316      psha
317      jsr      getCellOccupancy   ; Überprüft, ob die nächste Zelle
318      cmpa     player             ; den Stein des Spielers enthält.
319      pula
320      bne      getLineLengthLoopEnd
321
322      inx
323      cpx      #3                  ; Laufe nur 3 Steine lang
324      bne      getLineLengthLoop
325
326 getLineLengthLoopEnd
327
328      xgdx
329      tba
330      inca
331      bra      getLineLengthEnd   ; Inkrementiere A, sodass der neu
332                                   ; gelegte Stein mitgezählt wird
333 getLineLengthEnd
334
335      pulx
336      puly
337      pulb
338      rts
339
340
341 ; -----
342 ; isOutOfBorder:
343 ;
344 ; Gibt an ob die übergebenen Koordinaten sich innerhalb oder ausserhalb des
345 ; Spielfeldes befinden.
346 ;
347 ; @param Akku a: X-Kooridnate des Spielfeldes
348 ; @param Akku b: Y-Kooridnate des Spielfeldes
349 ; @return Akku a: 0 = innerhalb des Spielfeldes, 1 = ausserhalb des Spielfeldes
350 ; -----
351
352 isOutOfBorder
353
354      pshb
355
356      cmpa     #-1
```

### 3 Listening

```
357         beq     isOutOfBorderTrue
358         cmpa    #7
359         beq     isOutOfBorderTrue
360
361         cmpb    #-1
362         beq     isOutOfBorderTrue
363         cmpb    #6
364         beq     isOutOfBorderTrue
365
366         bra     isOutOfBorderFalse
367
368 isOutOfBorderTrue
369         ldaa    #1
370         bra     isOutOfBorderEnd
371
372 isOutOfBorderFalse
373         ldaa    #0
374         bra     isOutOfBorderEnd
375
376 isOutOfBorderEnd
377
378         pulb
379         rts
380
381 ; -----
382 ; playerwon:
383 ;
384 ; Gibt an ob die übergebenen Koordinaten sich innerhalb oder ausserhalb des
385 ; Spielfeldes befinden.
386 ;
387 ; @param Akku a: X-Kooridnate des Spielfeldes
388 ; @param Akku b: Y-Kooridnate des Spielfeldes
389 ; @return Akku a: 0 = innerhalb des Spielfeldes, 1 = ausserhalb des Spielfeldes
390 ; -----
391 playerwon
392
393 ; Flag setzen das die Eingabe blockiert und nur Reset erlaubt
394 jsr     LCDC1r
395 jsr     drawBoard
396 jsr     drawWinner
397
398     rts
```

### 3 Listening

Listing 3.7: Utils.inc

```
1      switch RomSection
2  ; -----
3  ; Utils.inc:
4  ;
5  ; Hilfsmethoden für das Terminal und den RAM.
6  ; -----
7
8
9  ; -----
10 ; clrRam:
11 ;
12 ; Setzt alle Bits im RAM auf 0.
13 ; -----
14
15 clrRam
16     psha
17     pshx
18
19     ldaa    #0
20     ldx     #ramBegin
21
22 clrRamLoop
23     staa    0,x
24     inx
25     cpx     #ramEnd
26     bne     clrRamLoop
27
28     pulx
29     pula
30     rts
31
32
33 ; -----
34 ; putChar:
35 ;
36 ; Zeichen in A auf dem Terminal ausgeben.
37 ;
38 ; @param Akku a: Das auszugebende Zeichen
39 ; -----
40
41 putChar    brclr scsr, #$80, * ; Warte, bis der Ausgabepuffer leer ist
42            staa scdr           ; Zeichen in A abschicken
43            rts
44
45
46 ; -----
47 ; crlf:
48 ;
49 ; Zeilenumbruch auf dem Terminal ausgeben.
50 ; -----
```

### 3 Listening

```
51
52 crlf      psha
53          ldaa #13          ; CR
54          jsr putChar
55          ldaa #10          ; LF
56          jsr putChar
57          pula
58          rts
59
60
61 ; -----
62 ; space:
63 ;
64 ; Leerzeichen auf dem Terminal ausgeben.
65 ; -----
66
67 space     psha
68          ldaa #32          ; Leerzeichen
69          jsr putChar
70          pula
71          rts
```

### 3 Listening

Listing 3.8: Trainer11Register.inc

```

1  ;*** Registerdefinitionen für den trainer11
2
3  PIO_A      equ    $8000
4  PIO_B      equ    $8001
5  PIO_C      equ    $8002
6  PIO_CONFIG equ    $8003
7
8  .ifndef MAP_PAGE
9  MAP_PAGE   equ    $00      ; wahlweise Offsets oder Adressen
10 .endif
11
12 ;*** Register ***
13
14 PORTA      equ    $00+MAP_PAGE ; Port A Data Register
15
16 PIOC       equ    $02+MAP_PAGE ; Parallel I/O Control Register
17 PORTC      equ    $03+MAP_PAGE ; Port C Data Register
18 PORTB      equ    $04+MAP_PAGE ; Port B Data Register
19 PORTCL     equ    $05+MAP_PAGE ; Alternate Latched Port C
20
21 DDRC       equ    $07+MAP_PAGE ; Port C Data Direction Register
22 PORTD      equ    $08+MAP_PAGE ; Port D Data Register
23 DDRD       equ    $09+MAP_PAGE ; Port D Data Direction Register
24 PORTE      equ    $0A+MAP_PAGE ; Port E Data Register
25 CFORC      equ    $0B+MAP_PAGE ; Timer Compare Force Register
26 OC1M       equ    $0C+MAP_PAGE ; Action Mask Register
27 OC1D       equ    $0D+MAP_PAGE ; Action Data Register
28 TCNT       equ    $0E+MAP_PAGE ; Timer Counter Register
29 TIC1       equ    $10+MAP_PAGE ; Timer Input Capture
30 TIC2       equ    $12+MAP_PAGE ; Timer Input Capture
31 TIC3       equ    $14+MAP_PAGE ; Timer Input Capture
32 TOC1       equ    $16+MAP_PAGE ; Timer Output Compare
33 TOC2       equ    $18+MAP_PAGE ; Timer Output Compare
34 TOC3       equ    $1A+MAP_PAGE ; Timer Output Compare
35 TOC4       equ    $1C+MAP_PAGE ; Timer Output Compare
36 TOC5       equ    $1E+MAP_PAGE ; Timer Output Compare
37 TCTL1      equ    $20+MAP_PAGE ; Timer Control Register 1
38 TCTL2      equ    $21+MAP_PAGE ; Timer Control Register 2
39 TMSK1      equ    $22+MAP_PAGE ; Timer Interrupt Mask Register 1
40 TFLG1      equ    $23+MAP_PAGE ; Timer Interrupt Flag Register 1
41 TMSK2      equ    $24+MAP_PAGE ; Timer Interrupt Mask Register 2
42 TFLG2      equ    $25+MAP_PAGE ; Timer Interrupt Flag Register 2
43 PACTL      equ    $26+MAP_PAGE ; Pulse Accumulator Control Register
44 PACNT      equ    $27+MAP_PAGE ; Pulse Accumulator Count Register High
45 SPCR       equ    $28+MAP_PAGE ; SPI Control Register
46 SPSR       equ    $29+MAP_PAGE ; SPI Status Register
47 SPDR       equ    $2A+MAP_PAGE ; SPI Data Register
48 BAUD       equ    $2B+MAP_PAGE ; SCI Baud Rate Register
49 SCCR1      equ    $2C+MAP_PAGE ; SCI Control Register 1
50 SCCR2      equ    $2D+MAP_PAGE ; SCI Control Register 2

```

### 3 Listening

```

51 SCSR      equ $2E+MAP_PAGE    ; SCI Status Register
52 SCDR      equ $2F+MAP_PAGE    ; SCI Data Register
53 ADCTL     equ $30+MAP_PAGE    ; A/D Control Register
54 ADR1      equ $31+MAP_PAGE    ; A/D Converter Result Register 1
55 ADR2      equ $32+MAP_PAGE    ; A/D Converter Result Register 2
56 ADR3      equ $33+MAP_PAGE    ; A/D Converter Result Register 3
57 ADR4      equ $34+MAP_PAGE    ; A/D Converter Result Register 4
58
59 OPTION     equ $39+MAP_PAGE    ; System Configuration Options
60 COPRST     equ $3A+MAP_PAGE    ; Arm/Reset COP Timer register
61 PPROG      equ $3B+MAP_PAGE    ; EEPROM Programming Control Register
62 HPRI0      equ $3C+MAP_PAGE    ; Highest Priority Interrupt Register
63 INIT       equ $3D+MAP_PAGE    ; RAM and I/O Mapping
64 TEST1      equ $3E+MAP_PAGE    ; Factory TEST Control Register
65 CONFIG     equ $3F+MAP_PAGE    ; COP, ROM and EEPROM Enables
66
67 ;*** Vektoren ***
68
69 VSCI       equ $FFD6
70 VSPI       equ $FFD8
71 VPAI       equ $FFDA
72 VPA0       equ $FFDC
73 VTOF       equ $FFDE
74 VTIC4      equ $FFE0
75 VTOC4      equ $FFE2
76 VTOC3      equ $FFE4
77 VTOC2      equ $FFE6
78 VTOC1      equ $FFE8
79 VTIC3      equ $FFEA
80 VTIC2      equ $FFEC
81 VTIC1      equ $FFEE
82 VRTI       equ $FFF0
83 VIRQ       equ $FFF2
84 VXIRQ      equ $FFF4
85 VSWI       equ $FFF6
86 VIOT       equ $FFF8
87 VCOP       equ $FFFA
88 VCLK       equ $FFFC
89 VRESET     equ $FFFE

```

## Abbildungsverzeichnis

1.1	Spieler . . . . .	2
1.2	Spielfeld . . . . .	3
1.3	Spielfeld . . . . .	4
1.4	Zellenbelegung . . . . .	5
1.5	Geany Konfiguration . . . . .	5
1.6	Geany Konfiguration . . . . .	6
1.7	Realterm . . . . .	7
2.1	Darstellung einer Zelle im RAM . . . . .	8
2.2	Darstellung einer leeren Zelle im RAM . . . . .	9
2.3	Darstellung einer Zelle mit einem Spielstein von Spieler 1 im RAM	10
2.4	Darstellung einer Zelle mit einem Spielstein von Spieler 2 im RAM	11
2.5	Programmablaufplan: <i>readInput</i> . . . . .	13
2.6	CAPTION . . . . .	16
2.7	CAPTION . . . . .	17
2.8	Zellenbelegung . . . . .	18
2.9	Spielerwechsel . . . . .	19
2.10	Zusammenhängend Steine . . . . .	20

## **Tabellenverzeichnis**