Mikroprozessor Workshop
Wintersemester 2019/2020

# Vier Gewinnt auf dem Motorola 68HC11 Prozessor

**Benutzer- und Programmierhandbuch**

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

13. April 2020, Hamburg

# Inhaltsverzeichnis

# 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 1.3), in das die Spieler abwechselnd ihre Spielsteine fallen lassen. Jeder Spieler besitzt 21 gleichfarbige Spielsteine (Spieler 1 komplett gefüllt, Spieler 2 innen leer. (Abbildung 1.2))

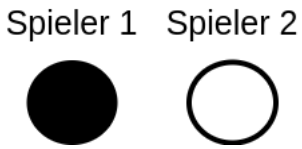


Abbildung 1.1: Spieler

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.

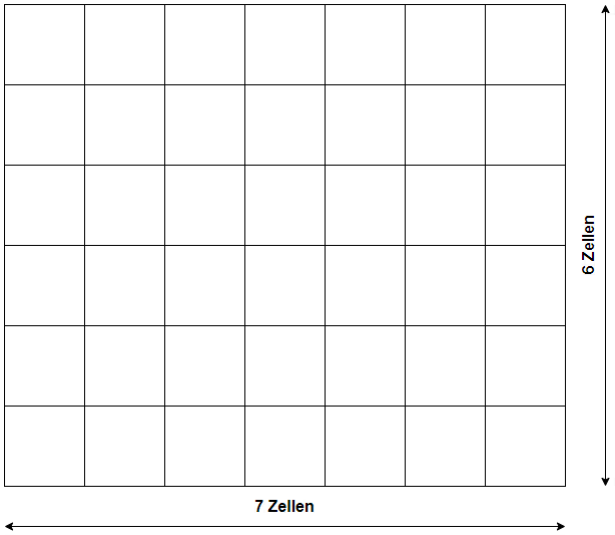


Abbildung 1.2: Spielfeld

## 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.

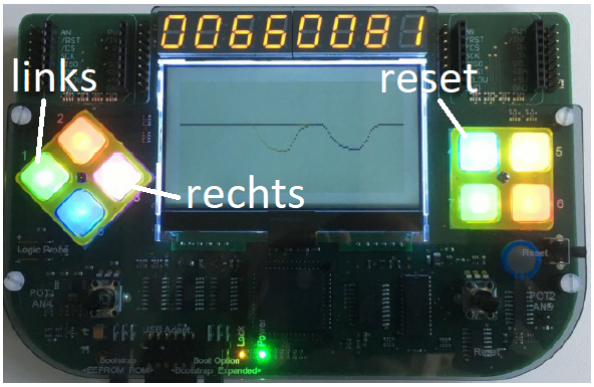


Abbildung 1.3: Spielfeld

## 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 1.5).

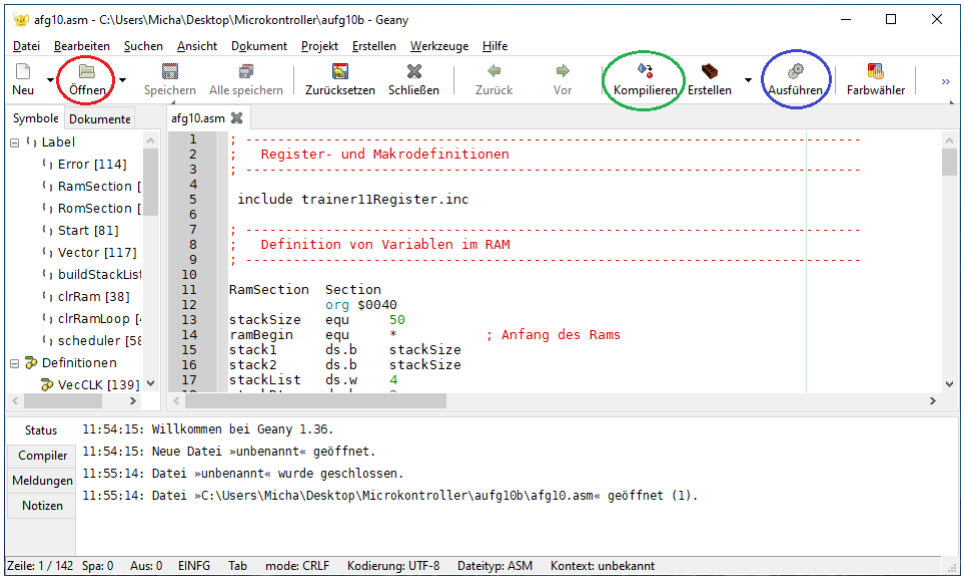


Abbildung 1.4: Zellenbelegung

Als nächstes müssen die Einstellungen in Geany geändert werden: Zunächst der verwendete Kompiler, welchen wir von miniIDE nutzen, welche vorher im Verzeichnis `C:\miniide` installiert sein muss (ansonsten muss der Pfad zum Kompiler 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 1.5).

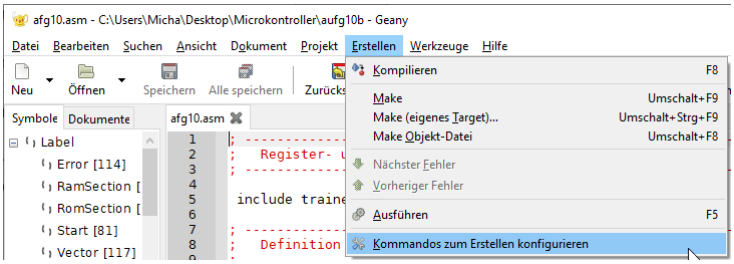


Abbildung 1.5: Geany Konfiguration

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"
```

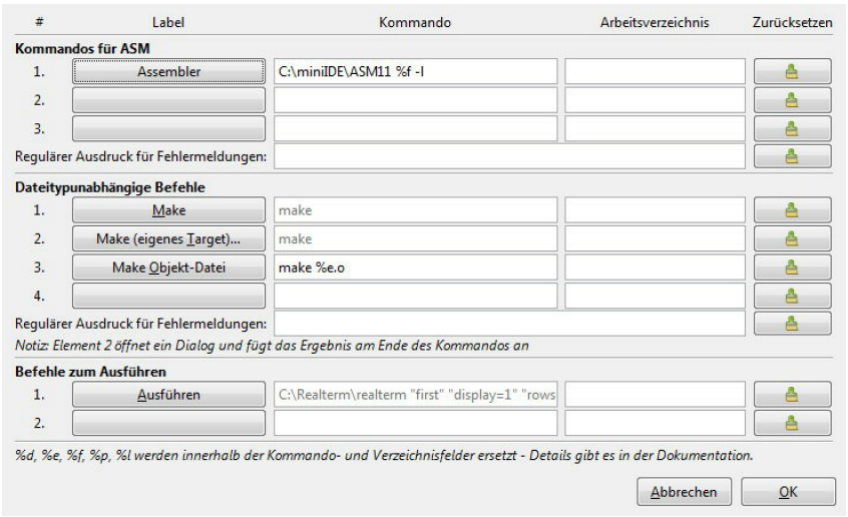


Abbildung 1.6: Geany Konfiguration

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 1.5). Nach dem Klick auf *Ausführen* öffnet sich jetzt *Realterm* (Abbildung 1.5). Nach betätigung des Resetschalters können die Daten mit *Send File*, im Reiter *Send*, ans Board übertragen werden.

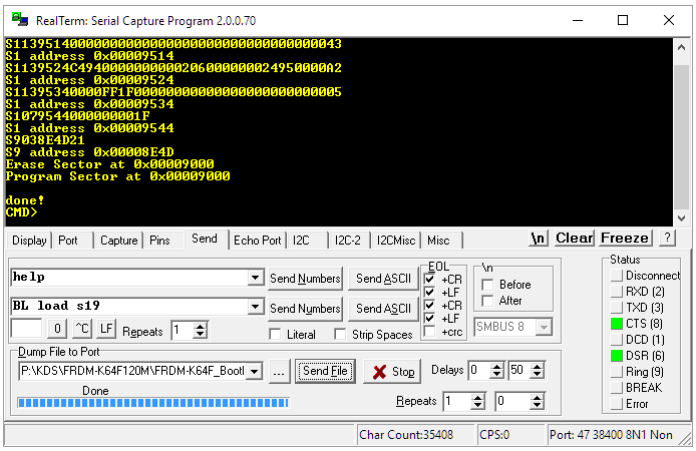


Abbildung 1.7: Realterm

# 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:

$$64Pixel = 8Pixel \cdot 8Pixel \tag{2.1}$$

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 linke Ecke. Um ein Pixel anzusteuern, wird das jeweilige Bit auf *1* bzw. auf *0* gesetzt.

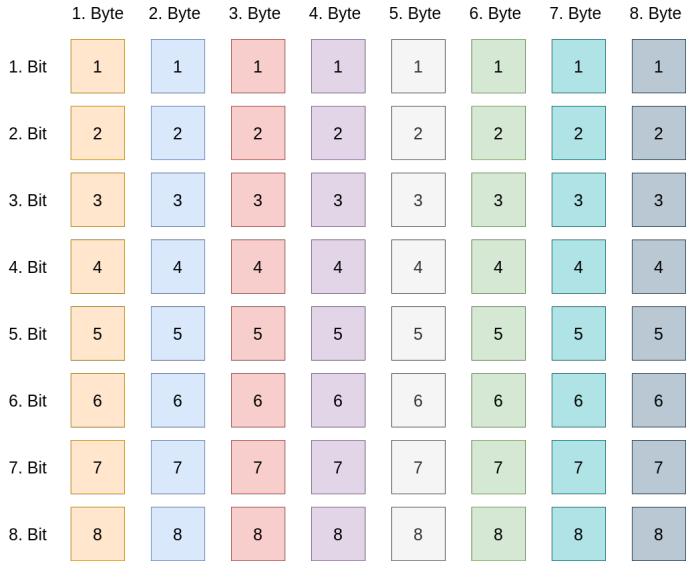


Abbildung 2.1: Darstellung einer Zelle im RAM

### 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.

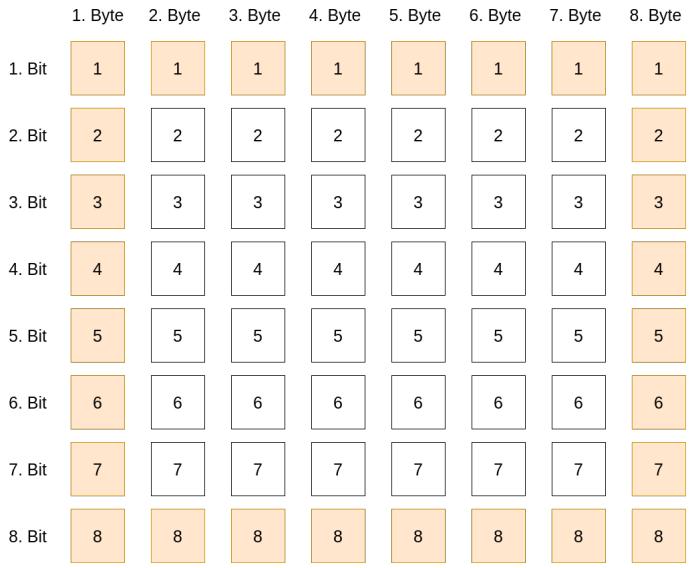


Abbildung 2.2: Darstellung einer leeren Zelle im RAM

### Spieler 1 Zelle

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

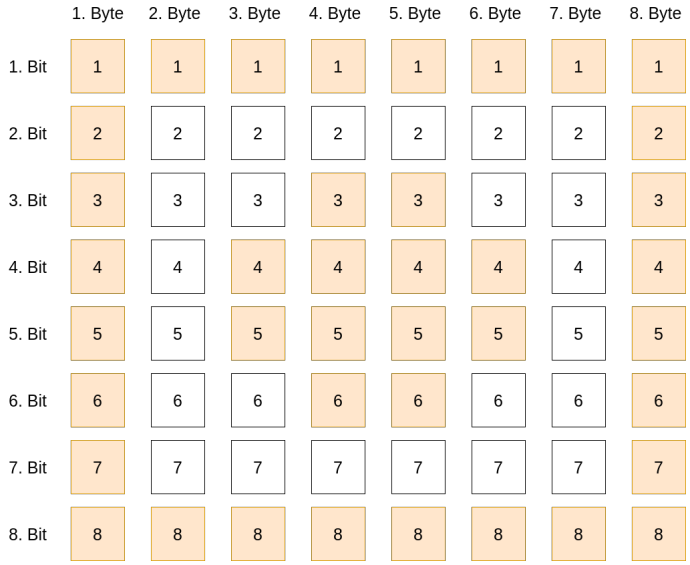


Abbildung 2.3: Darstellung einer Zelle mit einem Spielstein von Spieler 1 im RAM

### Spieler 2 Zelle

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

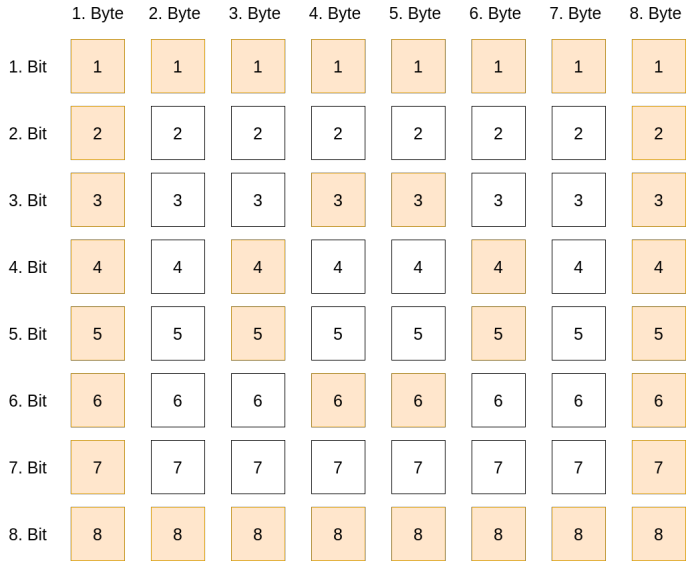


Abbildung 2.4: Darstellung einer Zelle mit einem Spielstein von Spieler 2 im RAM

### 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:

$$Buffergrösse = Zeilen \cdot Spalten \cdot Zellengrösse \tag{2.2}$$

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

$$336Byte = 6 \cdot 7 \cdot 8Byte \tag{2.3}$$

## 2.2 Eingabe

In dem folgenden Kapitel wird die Eingabe durch Tastendruck, deren Entprelleung und Flankenerkennung erläutet.

### 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 umd ein Entprellen der Tasten zu gewährleisten.

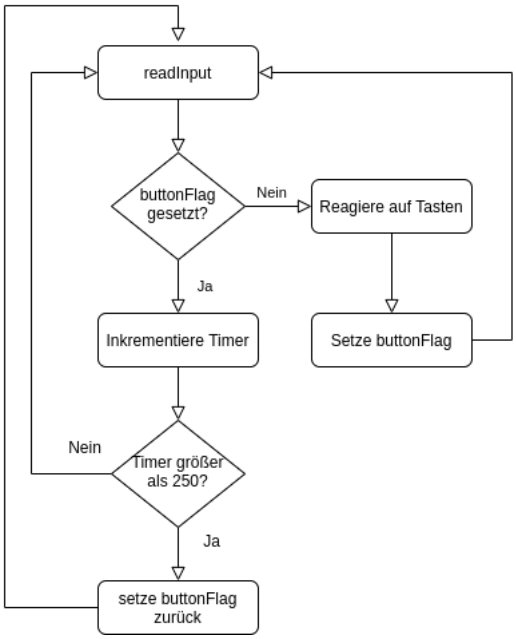


Abbildung 2.5: Programmablaufplan: *readInput*

## 2.3 Ausgabe

In diesem Kapitel wird erläutert wie die Ausgabe auf dem LCD realisiert wurde. Dabei werden die eigenheiten 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:

$$boardOffset = \frac{Displaybreite}{2} - \frac{Spielfeldbreite}{2} \tag{2.4}$$

Konkret bedeutet das:

$$36 = \frac{128Pixel}{2} - \frac{7Zeilen \cdot 8Pixel}{2} \tag{2.5}$$

### 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*:

```
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).

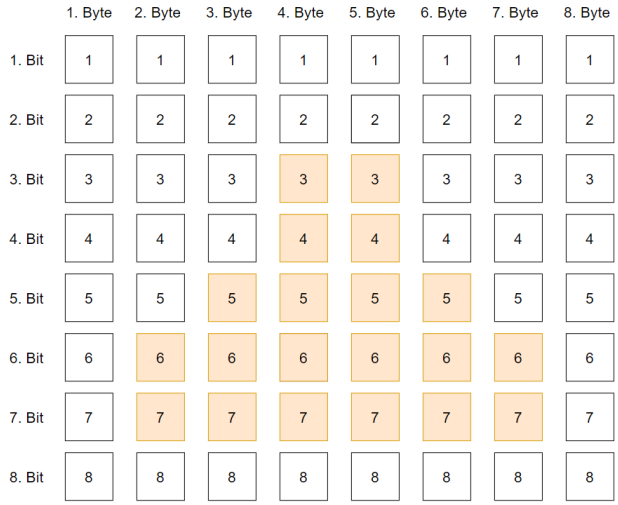


Abbildung 2.6: CAPTION

### 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

zu ermöglichen (Abbildung 8). Bei Versersetzen 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.

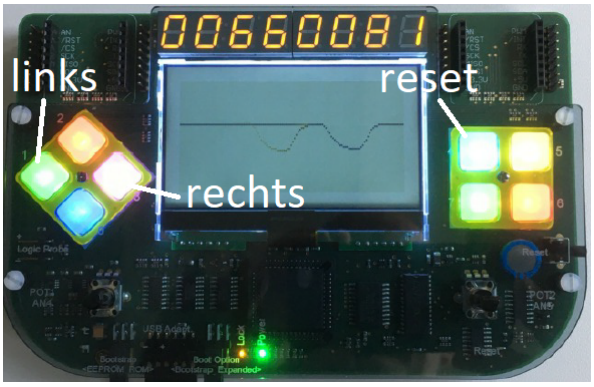


Abbildung 2.7: CAPTION

## 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 Steinfolge 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:

$$Addresse = (Spalte \cdot 8) + (Zeile \cdot 56) \tag{2.6}$$

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 währe die Berechnung folgende:

$$136 = (3 \cdot 8) + (2 \cdot 56) \tag{2.7}$$

### 2.5.2 Prüfen des Zelleninhalts

Um Feststellen zu können ob ein Spieler eine zusammenhängende Steinfolge 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.

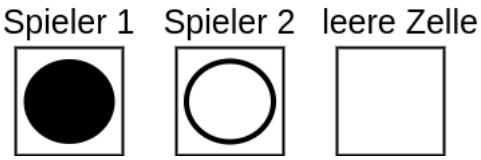


Abbildung 2.8: Zellenbelegung

### 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.

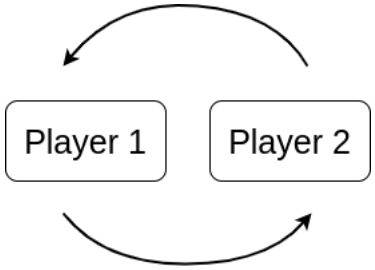


Abbildung 2.9: Spielerwechsel

### 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.

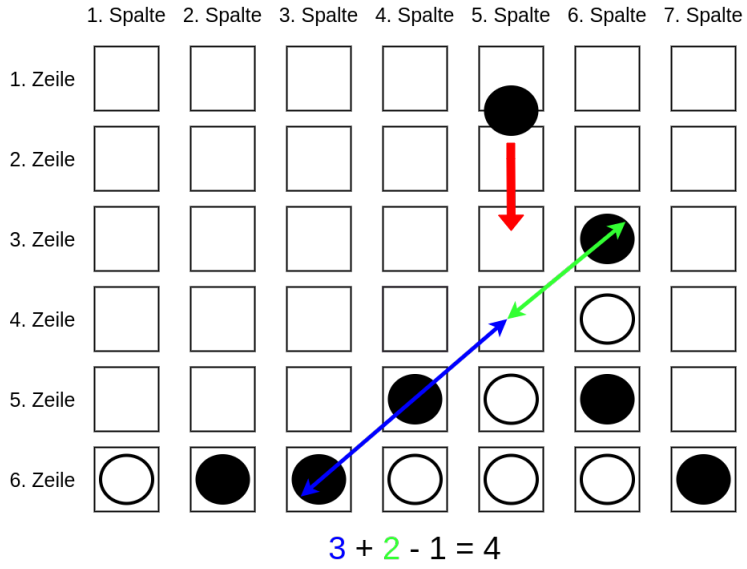


Abbildung 2.10: Zusammenhängend Steine

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

```
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
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             ; Anzahl an horizontalen Pixeln des Displays
74  LCDRows             equ         8               ; Anzahl an vertikalen Bytes des Displays
75  boardSize           equ         336             ; Byteanzahl des Buffers
76  byteSize            equ         8               ; Größe eines Bytes in Bit
77  rowLength           equ         56              ; Länge einer Zeile des Spielfeldes in Pixel
78  cursorRow           equ         6               ; Vertikale Position des Cursors in Byte
79  boardOffset         equ         36              ; Der horizontale Versatz des Spielfeldes
80  boardMiddleColumn   equ         24              ; Die mittlere Spalte des Spielfeldes
81
82
83  ;-------------------------------------------------------------------------------
84  ; Konstanten für Textausgabe auf dem LCD
85  ; der erste Wert jeder Konstante repräsentiert deren Länge in Byte
86  ;-------------------------------------------------------------------------------
87
88  turnText
89          dc.b        21,$02,$7E,$02,$00  ; T
90          dc.b        $3C,$40,$40,$7C,$00 ; u
91          dc.b        $7C,$08,$04,$08,$00 ; r
92          dc.b        $7C,$08,$04,$78,$00 ; n
93          dc.b        $28,$00             ; :
94
95  playerText
```

```
96          dc.b        30,$7E,$12,$12,$0C,$00  ; P
97          dc.b $02,$7E,$00                    ; l
98          dc.b $20,$54,$54,$78,$00            ; a
99          dc.b $1C,$A0,$A0,$7C,$00            ; y
100         dc.b $38,$54,$54,$48,$00            ; e
101         dc.b $7C,$08,$04,$08,$00            ; r
102         dc.b $00,$00                        ;
103
104 oneText
105         dc.b 4,$04,$7E,$00,$00              ; 1
106
107 twoText
108         dc.b 5,$64,$52,$52,$4C,$00          ; 2
109
110 wonText
111         dc.b 17,$00,$00,$3C,$40,$20
112         ,$40,$3C,$00    ; w
113         dc.b $38,$44,$44,$38,$00                ; o
114         dc.b $7C,$08,$04,$78,$00                ; n
115
116
117 ;-------------------------------------------------------------------------------
118 ;   Hauptprogramm
119 ;-------------------------------------------------------------------------------
120
121 initGame
122
123         psha
124
125         ldaa        #$0C                ; Löscht das Terminal
126         jsr         putChar
127         jsr         clrRam              ; Überschreibt den Ram mit Nullen
128         jsr         initSPI             ; Initialisierung der SPI-Schnittstelle
129         jsr         initLCD             ; Initialisierung des Displays
130         jsr         LCDClr              ; Löscht das Display
131         jsr         createGrid          ; Schreibt das Spielfeld in den Buffer
132         jsr         drawBoard           ; Schreibt den Bufferinhalt auf den Display
133         jsr         resetCursor         ; Setz den Cursor in die Mitte des Spielfeldes
134         ldaa        #1
135         staa        player              ; wechselt zu Spieler 1
136         jsr         showText
137         jsr         drawPlayer
138         pula
139
140         rts
141
142 Start
143         lds         #$3FFF              ; Einstiegspunkt des Spieles
144         jsr         initGame
145
146 mainLoop                                ; Hauptschleife
```

```
147         jsr         readInput
148         bra         mainLoop
149
150 End     bra         *
151
152
153 ;-------------------------------------------------------------------------------
154 ;   Vektortabelle
155 ;-------------------------------------------------------------------------------
156 VSection              Section
157         org         VRESET      ; Reset-Vektor
158         dc.w        Start       ; Programm-Startadresse
```

# Abbildungsverzeichnis

# Tabellenverzeichnis