Mikroprozessor Workshop
Wintersemester 2019/2020

Vier Gewinnt auf dem Motorola 68HC11 Prozessor

Benutzer- und Programmierhandbuch

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

26. März 2020, Hamburg

fhwedel
UNIVERSITY OF APPLIED SCIENCES

Inhaltsverzeichnis

1 Programmiererhandbuch

1.1 Spielfeld

1.1.1 Zelle

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

$$64Pixel = 8Pixel \cdot 8Pixel \tag{1.1}$$

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.

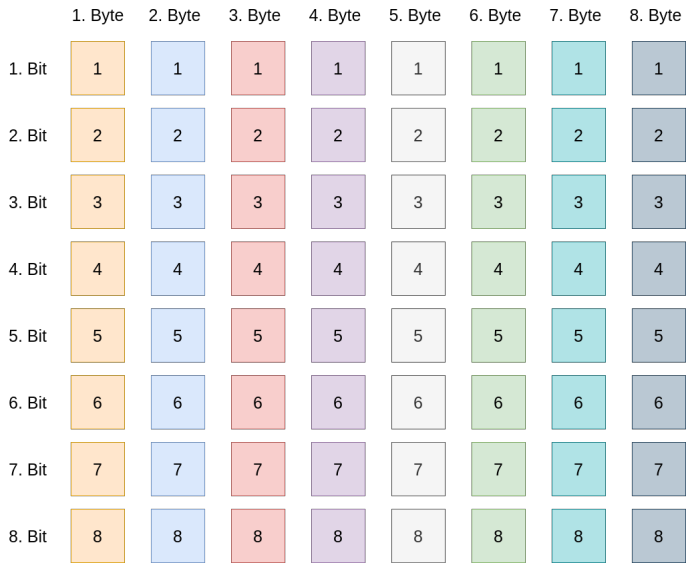

Abbildung 1.1: Darstellung einer Zelle im RAM

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.

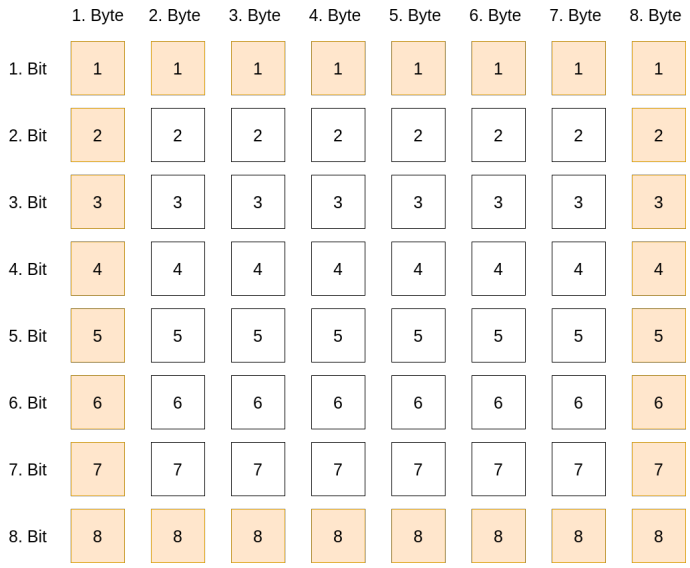

Abbildung 1.2: Darstellung einer leeren Zelle im RAM

Spieler 1 Zelle

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

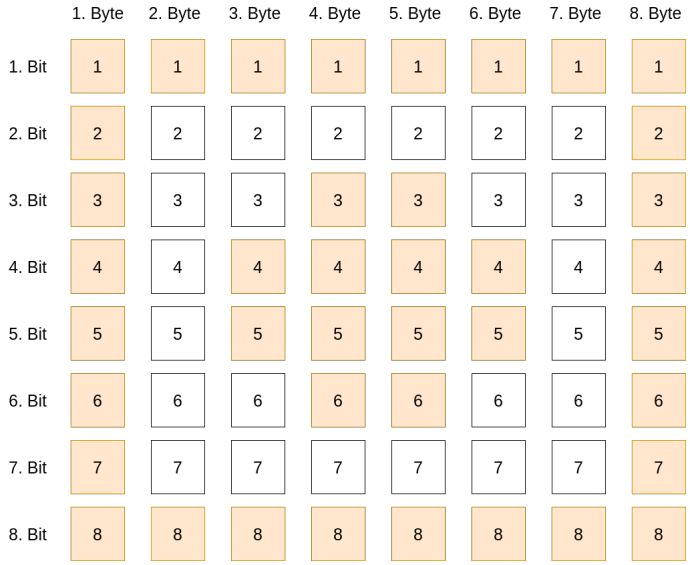

Abbildung 1.3: Darstellung einer Zelle mit einem Spielstein von Spieler 1 im RAM

Spieler 2 Zelle

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

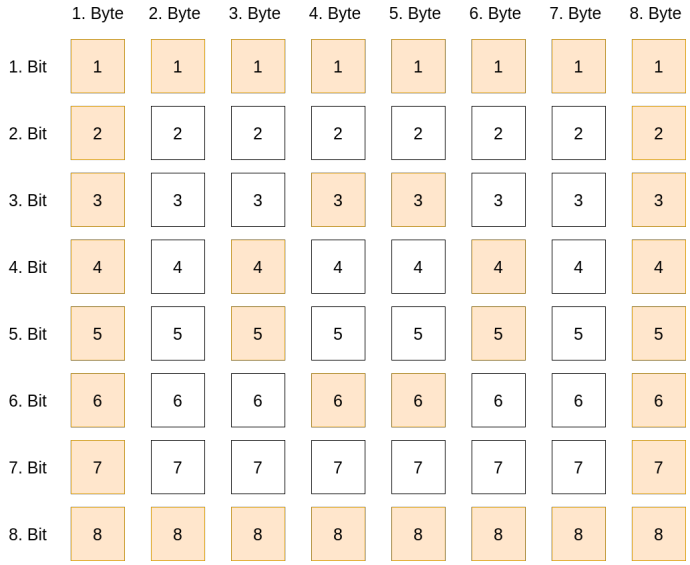

Abbildung 1.4: Darstellung einer Zelle mit einem Spielstein von Spieler 2 im RAM

1.1.2 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:

$$Buffergrösse = Zeilen \cdot Spalten \cdot Zellengrösse \tag{1.2}$$

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

$$336Byte = 6 \cdot 7 \cdot 8Byte \tag{1.3}$$

1.2 Eingabe

1.2.1 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.

1.2.2 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.

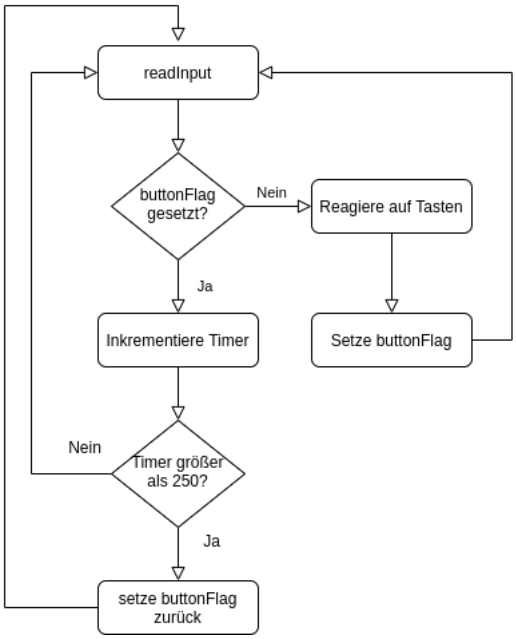

Abbildung 1.5: Programmablaufplan: *readInput*

Abbildungsverzeichnis

Tabellenverzeichnis