Mikroprozessor Workshop
Wintersemester 2019/2020

Vier Gewinnt auf dem Motorola 68HC11 Prozessor

Benutzer- und Programmierhandbuch

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

7. April 2020, Hamburg

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.

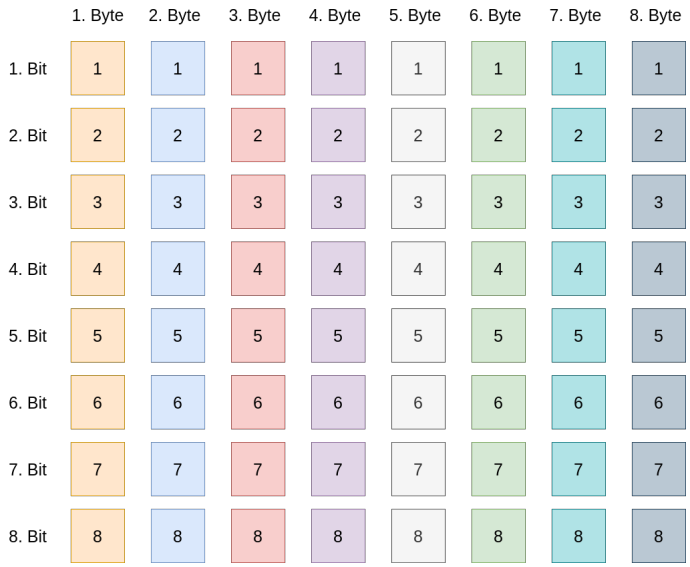

Abbildung 1.1: Darstellung einer Zelle im RAM

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.

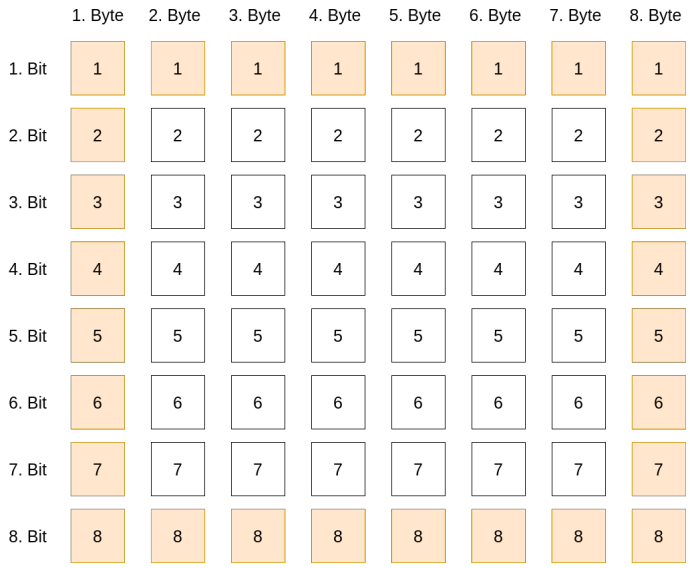

Abbildung 1.2: Darstellung einer leeren Zelle im RAM

Spieler 1 Zelle

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

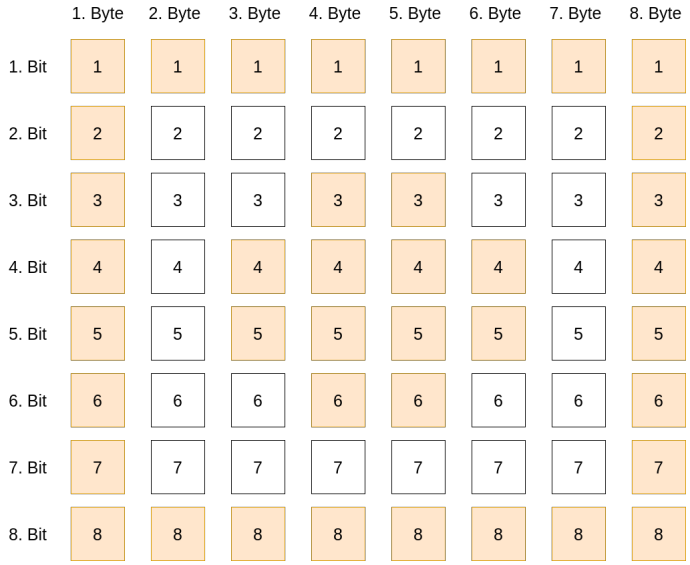

Abbildung 1.3: Darstellung einer Zelle mit einem Spielstein von Spieler 1 im RAM

Spieler 2 Zelle

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

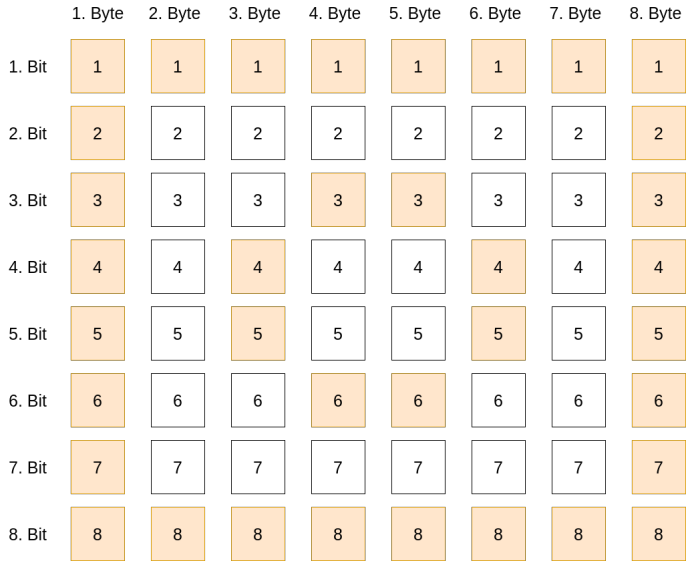

Abbildung 1.4: Darstellung einer Zelle mit einem Spielstein von Spieler 2 im RAM

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.

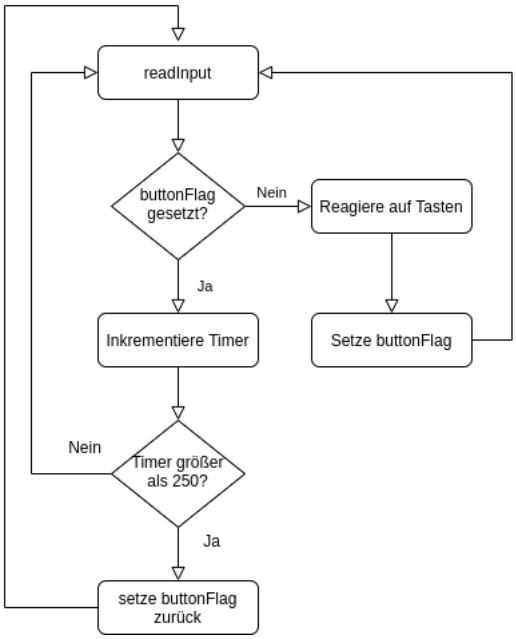

Abbildung 1.5: Programmablaufplan: *readInput*

1.3 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.

1.3.1 Aufbau

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

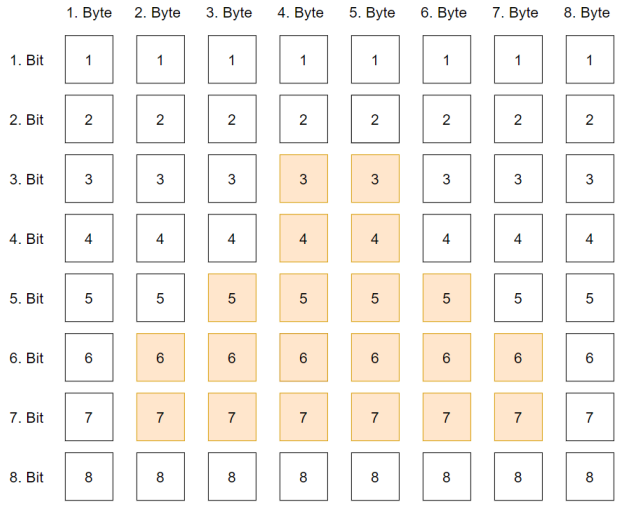

Abbildung 1.6: CAPTION

1.3.2 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.

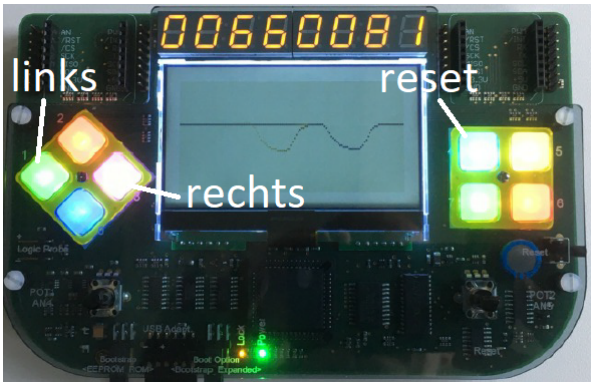

Abbildung 1.7: CAPTION

1.4 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.

1.4.1 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 die Spaltenzahl (1 – 7) mit 8 multipliziert und mit der Zeilenzahl (1 – 6) multipliziert mit 56 addiert. Also bei einer Spielfeldkoordinate bei Spalte 3 und Zeile 2 währe die Berechnung folgende: (3 * 8) + (2 * 56) = 136. Die Multiplikatoren ergeben sich aus einer Zellenhöhe von 8 für die Zeilenzahl und aus einer Zellenbreite von 8 bei 7 Zellen pro Zeile = 56.

1.4.2 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.

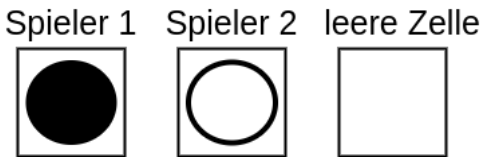

Abbildung 1.8: Zellenbelegung

1.4.3 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.

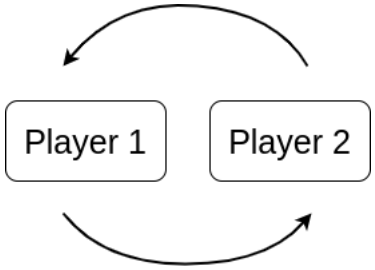

Abbildung 1.9: Spielerwechsel

1.4.4 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.

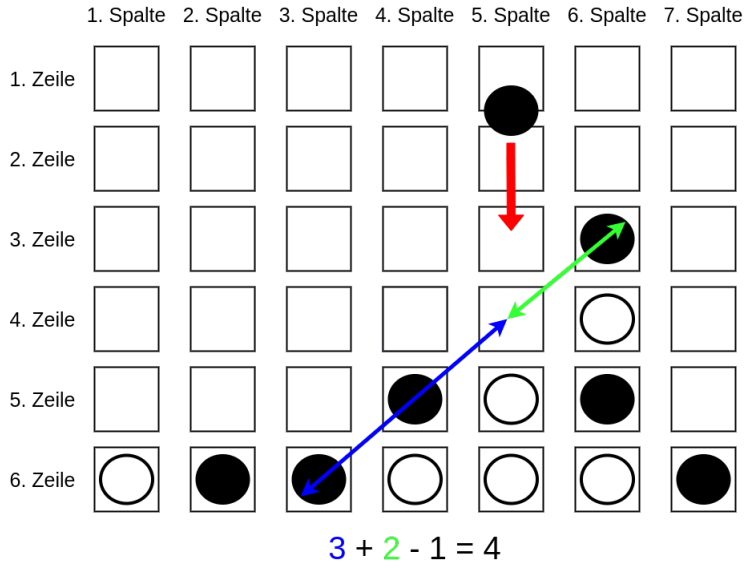

Abbildung 1.10: Zusammenhängend Steine

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.

Abbildungsverzeichnis

Tabellenverzeichnis