Systemnahe Programmierung

Game of Life

Projektdokumentation von Joshua Schulz, Raphael Müßeler, Kevin Hewener

Tinf17B2

24.05.2019

Inhalt

1. Einleitung

Heutzutage nimmt die Bedeutung von Embedded Systemen immer weiter zu. Unerlässlich für die effektive Programmierung dieser Systeme, ist Fachwissen im Bereich der hardwarenahen Programmierung.

In der heutigen Zeit ist es nicht mehr nötig über die Register und den exakten Befehlssatz des Prozessors, dank höheren Sprachen wie C oder Java Bescheid zu wissen. Für die Programmierung von Mikroprozessoren muss man allerdings oft Assembler beherrschen und den Aufbau der zu verwendeten CPU kennen.

Im Kurs „Systemnahe Programmierung“ haben wir uns Wissen in der Programmierung von Mikroprozessoren mithilfe von Assembler angeeignet und sie an diesem Projekt ausprobiert.

In den folgenden Kapiteln werden zuerst die verwendete Hardware und Entwicklungsumgebung vorgestellt. Danach wird dann unser Projekt beschrieben und die Implementierung erklärt.

1.1. Motivation

Persönlich haben wir noch nie Mikroprozessoren programmiert, weshalb sich die Suche nach einem möglichen Thema anfangs schwierig gestellt hat. Wir haben deshalb nachgedacht welche kleinen Programme wir schon in anderen höheren Sprachen programmiert haben. Schlussendlich haben wir uns für “Game of Life” geeinigt, da jeder aus unserer Gruppe es kannte und in anderen Programmiersprachen schon einmal programmiert hat.

1.2. Aufgabenstellung

Die Programmierung von “Game of Life” für den 8051 mithilfe von Assembler. “Game of Life” besteht aus drei Regeln die folgend lauten:

- Ein Lebewesen überlebt, wenn auf den 8 Nachbarfeldern zusammen 2 oder 3 Lebewesen existieren.
- Ein Lebewesen stirbt, wenn auf den 8 Nachbarfeldern zusammen weniger als 2 Lebewesen existieren (sterben an Isolierung) oder mehr als 3 Lebewesen existieren (sterben an Überbevölkerung).
- Ein Lebewesen wird auf ein leeres Feld geboren, wenn auf den 8 Nachbarfeldern zusammen genau 3 Lebewesen existieren.

2. Grundlagen

In den folgenden Kapiteln werden die verwendete Programmiersprache, IDE und der Mikrokontroller näher erläutert.

2.1. Assembler

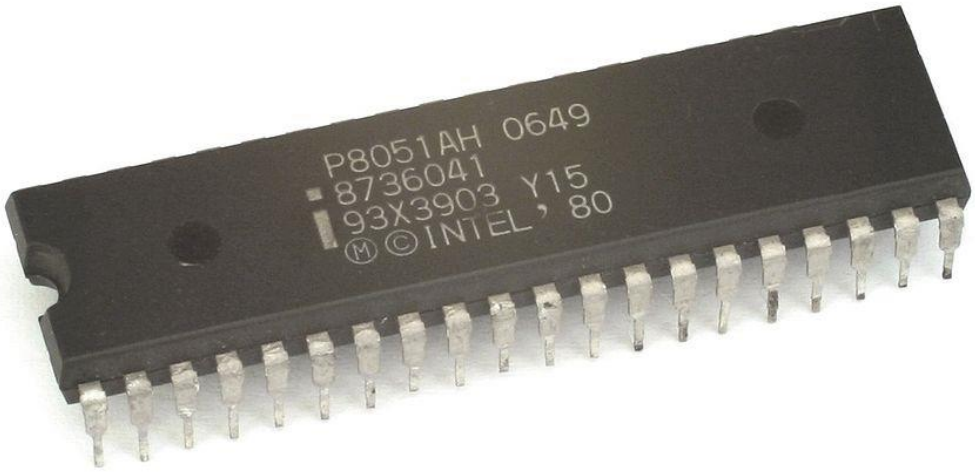
Assembler steht für eine Klasse von Programmiersprachen und deren zugehörigen Übersetzungsprogramme. Computer verstehen nur ihren eigenen Maschinencode. Das Programmieren in Maschinensprachen ist sehr fehleranfällig, unpraktisch und zeitraubend. Um Abhilfe zu schaffen wurden deshalb Mitte des 20. Jahrhunderts Assemblersprachen benutzt. Die so geschriebenen Programme müssen vor ihrer Ausführung erst in ein Maschinenprogramm übersetzt werden.

Alle Assemblersprachen sind stark maschinenabhängig, weshalb heutzutage höhere Programmiersprachen benutzt werden. Das einzige Einsatzgebiet für Assembler sind heutzutage Embedded Systems, da man in Assembler schnelle kompakte Programme schreiben kann.

2.2. Der 8051

Einer der heutzutage meistgenutzten Universalmikrokontroller ist der 8051 Mikrokontroller. Die große Familie der Familie MCS-51 ist verantwortlich für den Erfolg. Die 8051- oder besser MCS-51-Familie ist eine Prozessorarchitektur von Intel. Die verwendeten Chipsätze für die 8051-Familie sind von Herstellern wie Atmel, Philips, Infineon und Texas Instruments.

Mittlerweile ist der originale 8051 veraltet, aber es gibt viele weitere Varianten davon, von denen ein paar durchaus auf dem aktuellen Stand der Technik sind. Der 8051 hat 128 Byte internen RAM und kann jeweils bis zu 64 kB externer Daten- und Programmspeicher adressieren. Er besitzt 4 8-bit I/O Ports, zwei davon für den Zugriff auf externen Speicher. Er hat außerdem zwei externe Interrupts sowie 2 Timer/Counter.

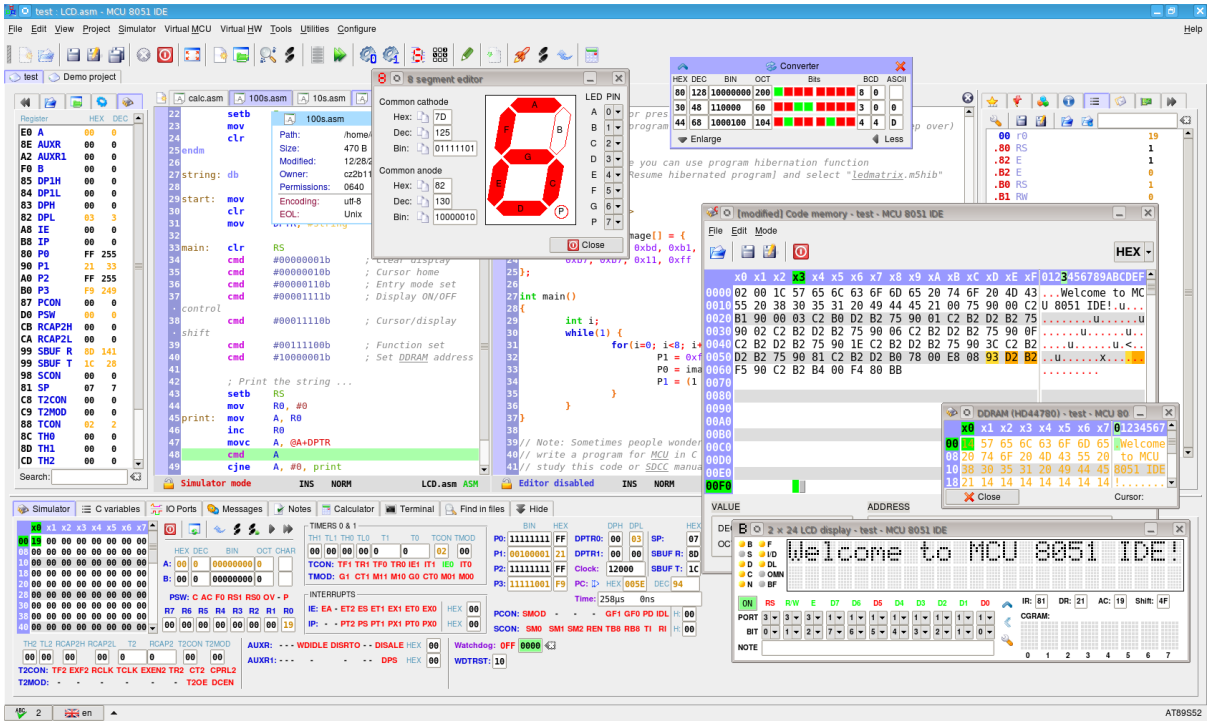


Der Original 8051 ist ein maskenprogrammierter Mikrocontroller. Es werden mindestens 12 Takte für einen Befehl benötigt. Befehls- und Datenspeicher werden über einen gemultiplexten externen Bus adressiert, wenn externe Speicher verwendet werden. Trotzdem sind beide logisch getrennt. Es ist nicht ersichtlich ob es eine Harvard-Architektur oder eine Von Neumann-Architektur ist.

Ein modernes 8051-Derivat wartet unter anderem neben DA-Wandler, I2C-Bus, höherer Taktfrequenz und AD-Wandler mit einer geringeren Taktteilung auf. Durch diese Anpassung werden nur noch 6, 4 oder nur 1 Takt für die Ausführung eines Befehls benötigt und nicht mehr 12. Ein externes EPROM ist durch einen internen Flash-ROM als Programmspeicher auch nicht mehr notwendig.

2.3. Entwicklungsumgebung

Die gewählte Entwicklungumgebung is dier IDE MCU 8051. Diese basiert auf den Prozessor Intel 8051. Die verwendete Entwicklungsumgebung unterstützt 2 Programmiersprachen, Assembler und C. Es werden viele “Debugging Features” unterstützt. Dazu zählt z.B register status, step by step, interrupt viewer, external memory viewer und code memory viewer. Sie ermöglicht die Simulation von verschiedenen Hardware-Komponenten wie z.B Displays, LEDs, Temperatursensoren und Tastern.


test · LCD.asm - MCU 8051 IDE
File Edit View Project Simulator Virtual MCU Virtual HW Tools Utilities Configure Help
8 segment editor
Converter
[modified] Code memory - test - MCU 8051 IDE
DDRAM (HD44780) - test - MCU 80
2 × 24 LCD display - test - MCU 8051 IDE
Welcome to MCU 8051 IDE!
Simulator
C variables
IO Ports
Messages
Notes
Calculator
Terminal
Find in files
Hide
Simulator mode
Editor disabled
AT89S52

3. Konzept

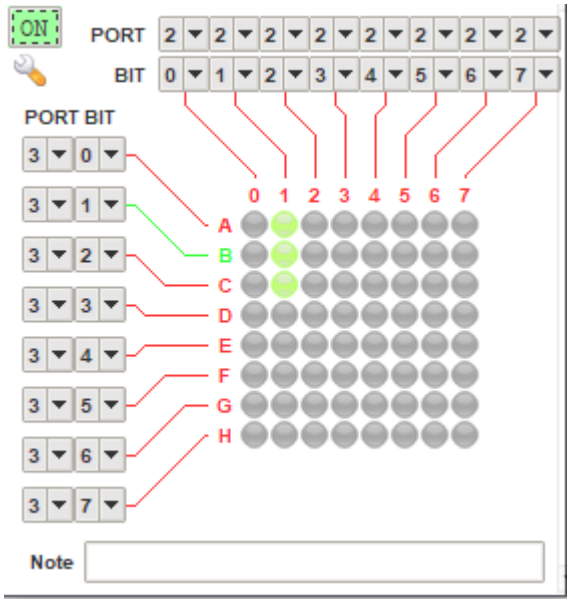
3.1. Analyse

Anforderungen

Für die Umsetzung von “Game of Life” wird keine Benutzereingabe benötigt. Die Initialbelegung ist im Programmcode festgelegt. Sobald das Programm gestartet ist läuft “Game of Life”. Die Ausgabe erfolgt mithilfe der LED Matrix.

Portbelegung

Der 8051 Mikrocontroller stellt für die Ein- und Ausgabe von Daten 4 Ports mit jeweils einem Byte zur Verfügung. Über diese Ports muss die oben genannte Ausgabe erfolgen. Für die Ausgabe der einzelnen Zellen werden die Ports 2 und 3 verwendet. Das nachfolgende Bild zeigt die Belegung der 8x8 LED Matrix.



3.2. Programmentwurf

Der Programmablauf ist in unterschiedliche Stufen gegliedert:

- Anzeigen der lebenden Zellen auf der 8x8 LED Matrix
- Berechnen der Anzahl der umliegenden lebenden Zellen (auch als lebenden Nachbarn bezeichnet)
- Den Zustand der aktuellen Zellen bestimmen

- Abhängig von der Anzahl der lebenden Nachbarn die aktuelle Zelle leben bzw. sterben lassen (gemäß der definierten Regeln des Conway's Game of Life)

Diese Stufen werden solange wiederholt, bis das Programm abgebrochen wird.

4. Implementierung

4.1. Anzeigen

Das Anzeigen der lebenden Zellen geschieht wie folgt: Alle lebenden Zellen werden im Hauptspeicher unter `020H` abgelegt. Dabei repräsentiert die gesamte Reihe im Hauptspeicher (also bis `027H`) alle anzuzeigenden Zellen und jede einzelne Adresse eine Reihe. Somit ergeben sich 8x8 Zellen.

Nun werden zeilenweise alle lebenden Zellen in der 8x8 LED Matrix angezeigt. Die 8x8 LED Matrix ist so konfiguriert, dass wenn an beiden Ports eine 0 anliegt, so leuchtet die entsprechende Zelle, Reihe oder Spalte.

Wenn also beispielsweise in der Adresse `020H` die Zahl `#0FEh` gespeichert ist (binär: `1111 1110`), ist die erste Zelle in der ersten Zeile lebend und wird somit angeschaltet.

4.2 Lebenden Nachbarn bestimmen

Um die lebenden Nachbarn zu bestimmen, wird zunächst ein Zähler benötigt. Nun wird bei allen umliegenden Zellen geprüft, ob diese lebendig (0) oder tot (1) sind. Ist der Wert der Zelle `#0FFh`, so liegt diese außerhalb der LED Matrix und wird somit übersprungen. Nachdem alle umliegenden Zellen geprüft wurden, steht die Zahl der lebenden Zellen in `R2`.

4.3 Anwenden der Regeln

Abhängig von der Zahl der lebenden Nachbarn können nun die Regeln des Game of Life angewandt werden. Hierbei wird zunächst unterschieden, ob die aktuelle Zelle lebt oder nicht. Ist dies entschieden (also an aktueller Position steht eine 0), so werden können die Regeln ausgeführt werden. Hierbei muss ausschließlich abhängig vom Zustand der aktuellen Zelle, sowie der Anzahl der lebenden Nachbarn, der Zustand des aktuellen Status geändert oder nicht geändert werden.

Das Anwenden der Regeln geschieht auf einer Arbeitskopie des aktuellen Status, sodass keine Konflikte entstehen können, wenn sich der aktuelle Status ändert.

So entsteht ein andauernder Prozess, der für die aktuelle Zelle zunächst die lebenden Nachbarn bestimmt und anschließend die Regeln anwendet.

5. Fazit

Die Entwicklung von Game of Life in Assembler hat gezeigt, dass der Einsatz das die hardwarenahe Programmierung nicht immer reibungslos funktioniert und dass das Erreichen einer gewünschten Funktionsweise viel Zeit beanspruchen kann. Durch das Programm konnten wir einen Einblick erhalten wie komplex und wieviel Aufwand selbst in einem vermeintlich kleinen Programm steckt, wenn dieses systemnah umgesetzt wird.

6. Literatur

https://www.mikrocontroller.net/articles/8051

https://de.wikipedia.org/wiki/Intel_MCS-51

https://www.mikrocontroller.net/articles/Assembler

https://en.wikipedia.org/wiki/MCU_8051_IDE