Vertiefung Prozessdatenverarbeitung Sommersemester 2008
Fachbereich Design Informatik Medien

Lichtwuerfel

Michael Frey
mail@mfrey.net

18. August 2008

Inhaltsverzeichnis

1 Einleitung

Das Projekt wurde im Rahmen der Vertiefung Prozessdatenverarbeitung entworfen und umgesetzt. Ziel des Projektes war der Entwurf und die Konstruktion eines dreidimensionalen Würfels aus 1000 Leuchtdioden, sowie das entwickeln der Steuersoftware und Beispielanwendungen für den Würfel. Die Leuchtdioden sind dabei in einem dreidimensionalen Gitter angeordnet. Jede LED soll einzeln ansteuerbar sein. Auf dem Würfel sollen Animationen abgespielt werden können und Prozesse visualisiert werden.

Der 3D Display Cube bestand aus einem Framework was es ermöglichte Photoshop Dateien einzulesen, Animationen zu bauen oder externe Datenquellen wie Kameras oder Audioquellen anzuschliessen. Verwandt mit dem 3D Display Cube ist das Cubatron Projekt der Network Wizards die das Konzept aufgeriffen und vergrösserten. Der Cubatron besteht aus 768 LEDs die in einem Gitter aus 2,5 x 2,5 x 2,5 Meter angeordnet sind. Die LEDs befinden sich beim Cubatron in angebohrten Tischtennisbällen. Das Konzept der angebohrten Tischtennisbällen wurde unter anderem auch in der regionalen Vertretung des Chaos Computer Clubs in Mainz aufgegriffen. Als Material für die Verstrebungen wurde damals Federstahl gewählt. Der Federstahl war der Grund warum bereits nach wenigen Wochen der Cube bereits kaputt war. Durch den starren Federstahl waren zwar die LEDs in einer relativ unflexiblen Gatter positioniert aber der Cube neigte sich leicht zur Seite was mit der Zeit dazu führte dass die Lötstellen brachen und der Cube sich sozusagen selbst zerlegte.

Das Vertiefungsprojekt war der Versuch einen neuen LED Cube, auf Basis der Erfahrungen des ersten LED Cubes des CCC Mainz, zu bauen und eine umfangreiche Anleitung zum nachbauen zur Verfügung zu stellen. Die Ausarbeitung für das Projekt gliedert sich in Grundlagen, Analyse, Implementierung, Bewertung und Ausblick.

2 Grundlagen

2.1 Serial Peripheral Interface

Das Serial Peripheral Interface (SPI) ist ein von Motorola entwickelter synchroner serieller Bus, wo digitale Schaltungen nach einem Master-Slave Prinzip miteinander verbunden werden. Jeder Teilnehmer in SPI ist an vier Leitungen angeschlossen:

- SCK (Serial Clock)
- MOSI (Master Out Slave In)
- MISO (Master In Slave Out)
- SS (Slave Select)

Zusätzlich wählt der Master den Slave über ein Select Slave (SS) Leitung aus. Der Slave wird aktiv und "lauscht" an der MOSI Leitung auf eingehende Daten. Ist die Übertragung beendet so wird in dem SPI Status Register (SPSR) das SPI Interrupt Flag (SPIF) gesetzt. Bei der Initialisierung der Pins ist unbedingt darauf zu achten dass der SS des Masters auf Output gesetzt wird, damit nicht aus Versehen der SS von einem Slave gesetzt wird (auch wenn das eher unwahrscheinlich ist).

2.2 Atmel Prozessoren

Für die Platinen wurde 8-Bit RISC Prozessoren von Atmel[1] eingesetzt. Die 8-Bit Prozessoren werden von Atmel unter anderem in der Produktreihe ATmega vertrieben. Der ATmega Prozessor zeichnet sich durch eine relativ einfache Handhabung aus. Über einen In System Programmer (ISP) werden die Prozessoren üblicherweise via USB-, Parallel- oder Seriellschnittstelle programmiert. Das Protokoll zur Übertragung der Daten über einen ISP ist in der Regel SPI. Im Gegensatz zu anderen MCUs ist es nicht nötig den ATmega aus der Zielschaltung zu entfernen um ihn programmieren zu können. Für den ATmega existiert eine Reihe von frei verfügbaren Compilern, Assembler, Linkern und

[1] http://www.atmel.com

Entwicklungsumgebungen. Das Speichermanagement des ATmegas entspricht dem Harvard Prinzip, das bedeutet dass es getrennte Adressräume für den Flash Speicher, das EEPROM und den RAM existieren. Für das Projekt wurde unter anderem vier ATmega8515 eingesetzt, die mit einem Takt von bis zu 16 Mhz arbeiten.

2.3 Arduino

Der Arduino ist eine Platine bestehend aus einer USB Schnittstelle, einer ATmega168 Micro Controller Unit, 14 digitalen IO Pins (wovon 6 Pins PWM unterstützen) und 6 analogen IO Pins. Das Layout der Platine ist frei verfügbar und verfolgt damit den Ansatz von Open Hardware. Jeder kann mit dem Schaltplan, sofern er über die entsprechenden Ressourcen verfügt, die Platine nachbauen. Programmiert wird der Arduino in der Arduino Programming Language die auf Wiring [2] basiert. Die Syntax der Sprache ist relativ ähnlich zu C. Die ebenfalls frei verfügbare Entwicklungsumgebung übersetzt den Code in der Arduino Programming Language in C++ Code. Mit der Nähe zu C/C++ und der letztendlichen Übersetzung des Codes in C++ Code ist es möglich Sprachelemente und Bibliotheken zu nutzen die in C/C++ geschrieben sind. Natürlich ist es auch möglich völlig unabhängig von der Arduino Programming Language und der entsprechenden Entwicklungsumgebung in anderen Sprachen den Arduino zu programmieren, sofern sie den ATmega168 unterstützen.

[2] http://wiring.org.co/

3 Analyse

3.1 SPI Protokoll

Für die Übertragung von Daten per SPI wurde ein eigenes Datenformat erdacht dass aus einem Header und einem Payload besteht. Der Header gibt an ob es sich bei dem Payload um ein Byte mit einer Ebenennummer handelt oder um 250 Byte Säulendaten. Sind es Ebenendaten so steht im Header ein 0xFF, ansonsten 0xEE. In Abbildung 3.1

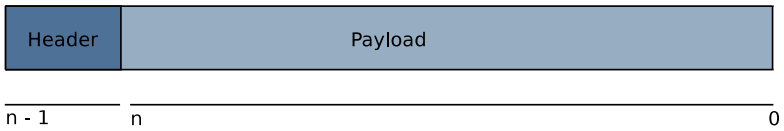

Abbildung 3.1: *SPI Paket*

ist das SPI Paketformat dargestellt. Sollte es zu Problemen führen 250 Byte Daten in einem Stück zu transportieren, so könnte man über eine zusätzliche Segmentierung der Daten nachdenken. Beispielsweise wäre es möglich nach dem Header ein zusätzliches Byte bei der Übertragung der Säulendaten einzuführen. Das zusätzliche Segmentierungsbyte würde es erlauben einen Art Multiplikator anzugeben, gefolgt von 10 Byte Nutzdaten. Sollen die Daten der Leuchtdioden von 20 bis 29 geändert werden, so würde das wie in Abbildung 3.2 zu sehen, aussehen. Im Beispiel wäre die LED 20 und die LED 29 jeweils

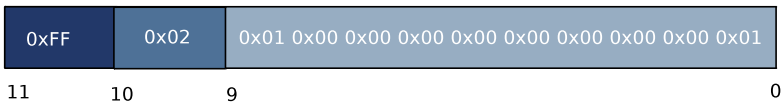

Abbildung 3.2: *Alternatives SPI Paket Beispiel*

an der entsprechenden Säule aus. An der zu übertragenden Menge der Daten würde sich mit dem alternativen Datenformat nichts ändern, allerdings würden die Schaltzeiten "entlastet" werden. Um einen Einfluss auf die Menge der zu übertragenden Daten zu gewinnen müsste man eine andere Codierung der Daten wählen oder über eine andere Interpretierung der zu übertragenden Daten nachdenken. Zum Beispiel ist es sicherlich

verbesserswert nur wirklich nötige Änderungen zu übertragen statt den kompletten Buffer auf dem Säulenprozessor zu überschreiben. Die Pakete werden natürlich "komplexer", bieten dafür aber auch mehr Möglichkeiten.

3.2 Leuchtdioden

Die Anoden aller Leuchtdioden einer Ebene sind miteinander verbunden. Indem man diese Anoden auf 0V oder +5V legt, kann man die Ebene deaktivieren bzw. aktivieren (die Anode muß auf 0V liegen, damit die Leuchtdiode leuchten kann). Weiterhin sind die Kathoden aller Leuchtdioden, die übereinander liegen (also jede in einer anderen Ebene), miteinander verbunden. Diese Leuchtdioden bilden zusammen eine Säule. Es gibt also 10 x 10 = 100 Säulen, und jede Säule hat 10 Leuchtdioden. Man kann jede Säule einzeln aktivieren/deaktivieren, indem man sie auf 5V/0V legt. Wenn man nun eine oder mehrere Säulen aktiviert, aber nur eine Ebene, dann leuchtet von jeder aktiven Säule genau die Leuchtdiode, die in der aktiven Ebene liegt. In Abbildung 3.3 ist eine verein-

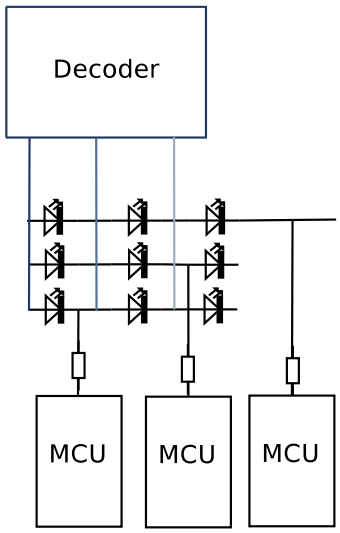

Abbildung 3.3: *Funktionsweise LED Ansteuerung*

fachte schematische Darstellung der Ansteuerung der LEDs zu sehen. Im Beispiel wird ein LED Cube aus 1 x 3 Leuchtdioden, also 9 Leuchtdioden, angesteuert. Der LED Cube ist eindimensional, was allerdings für das Modell und die prinzipielle Vorgehensweise unerheblich ist. Weiterhin besitzen die drei Prozessoren, im Bild als MCU bezeichnet, nur jeweils einen IO Port und der Decoder besitzt drei Ausgänge. Um die erste Ebene, im Bild angedeutet mit der dunkelblauen Linie, zu aktivieren muss der Decoderausgang

auf High, also +5 Volt, gezogen werden. Soll nun die zweite Leuchtdiode in der ersten Ebene leuchten, so muss am Ausgang des zweiten Prozessor der IO Pin auf Low, also 0 Volt, gezogen wird.

3.3 Steuerung

Die Ansteuerung der Säulen erfolgt über 4 ATmega 8515 die einzeln jeweils über 35 IO Ports verfügen, insgesamt 140. Die Kommunikation mit dem Arduino Board erfolgt über SPI. Es sind vier ATmega 8515 nötig da bereits 4 Pins pro Prozessor zur SPI Kommunikation vergeben sind (SCLK (Serial Clock), MOSI (Master out Slave in), MISO (Master in Slave out), SS (Slave Select)). Jeweils 4 Pins auf dem Arduino Board dienen als Slave Select und sind jeweils direkt mit dem SS Port des jeweiligen ATmega verbunden, damit weis jede MCU ob sie jetzt auf dem Bus nach Daten lauschen soll. Über einen Demultiplexer werden dann die Ebenen ausgewählt, wobei immer nur eine Ebene aktiv sein kann. Werden die Ebenen entsprechend schnell durchgeschaltet so entsteht der Eindruck eines stehenden Bildes.

4 Design

Im Kapitel werden die Schaltpläne für die Platinen vorgestellt und erklärt.

4.1 Ebenensteuerung

Die Ebenensteuerung besteht aus einem Decoder Baustein der mit vier Eingangssignale zehn Ausgangssignale erzeugt. Für Tests wurden auf die Platine zusätzliche Jumper angebracht um unterschiedliche Werte an den Eingängen ”manuell” setzen zu können. Vor den Eingängen des Decoders sind vier Pull Down Widerstände angebracht. Wenn die Eingänge des Decoders nicht mit High (5 Volt) oder Low (0 Volt) belegt sind, dass heisst der Arduino ist nicht angeschlossen und die Jumper sind nicht gesteckt, dann ziehen die Pull-Down-Widerstände die Eingänge auf 0V, damit sie nicht "flackern". Das verhindert, dass der Decoder zufällige"Werte decodiert. Die MOSFETs arbeiten mit einer inversen

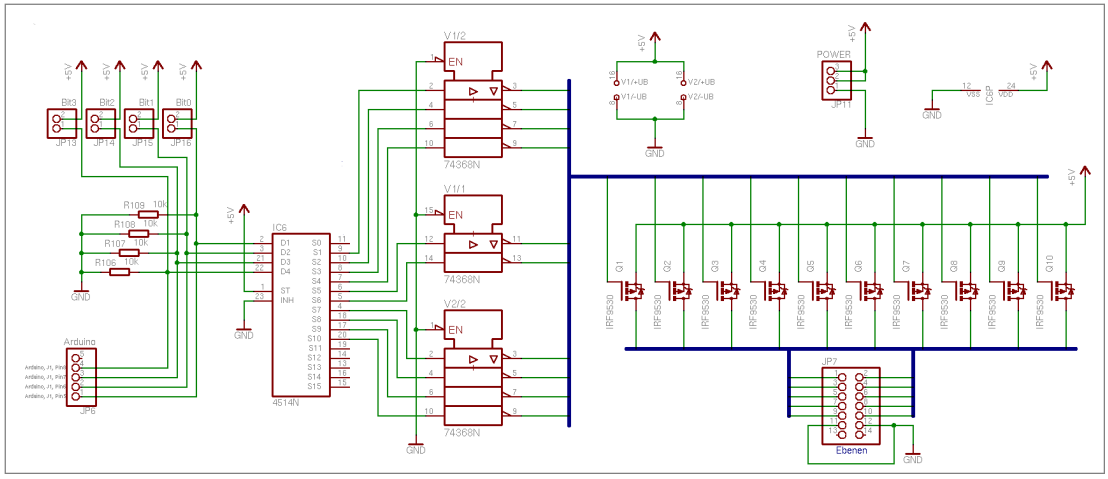

Abbildung 4.1: *Schaltplan der Ebenensteuerung*

Logik das bedeutet dass bei einer logischen 0, der MOSFET auf 5 Volt schaltet und bei einer 1 die Verbindung unterbricht. Da die LEDs der betroffenen Ebene dann mit einem Bein nicht angeschlossen sind, können sie auch nicht leuchten. Die Inverter vor den MOSFETs invertieren die Ausgangssignale des Decoders, damit die gewünschte Ebene auch

geschaltet wird. Prinzipiell wäre es möglich gewesen den Decoder direkt mit den Ebenen zu verbinden allerdings sind im "ungünstigsten" Fall maximal 100 LEDs an, wobei jede LED 20 mA vebraucht was zu einem Verbrauch von 2 A führt mit dem man den Decoder zerstören würde. MOSFETs haben zusätzlich den Vorteil, dass sie einen sehr geringen Widerstand haben, wenn sie die Ebenen mit 5V verbinden. Das bedeutet, dass nur wenig Spannung an ihnen abfällt und sie fast keine Wärme erzeugen.

4.2 Säulensteuerung

Die Säulensteuerung besteht aus vier ATmega8515 Prozessoren die jeweils mit 25 Pins den Säulen verbunden sind. In Bild 4.2 ist der Schaltplan der Säulensteuerung zu sehen. Zusätzlich ist auf der Platine als externer Taktgeber ein Oszillator verbaut der es erlaubt die Prozessoren mit einem höheren Takt zu betreiben. Neben den Prozessoren findet sich eine LED die das entwickeln erleichtern soll. Mit der LED kann gezielt geprüft werden ob ein bestimmter Programmzweig durchlaufen wird oder auch nicht.

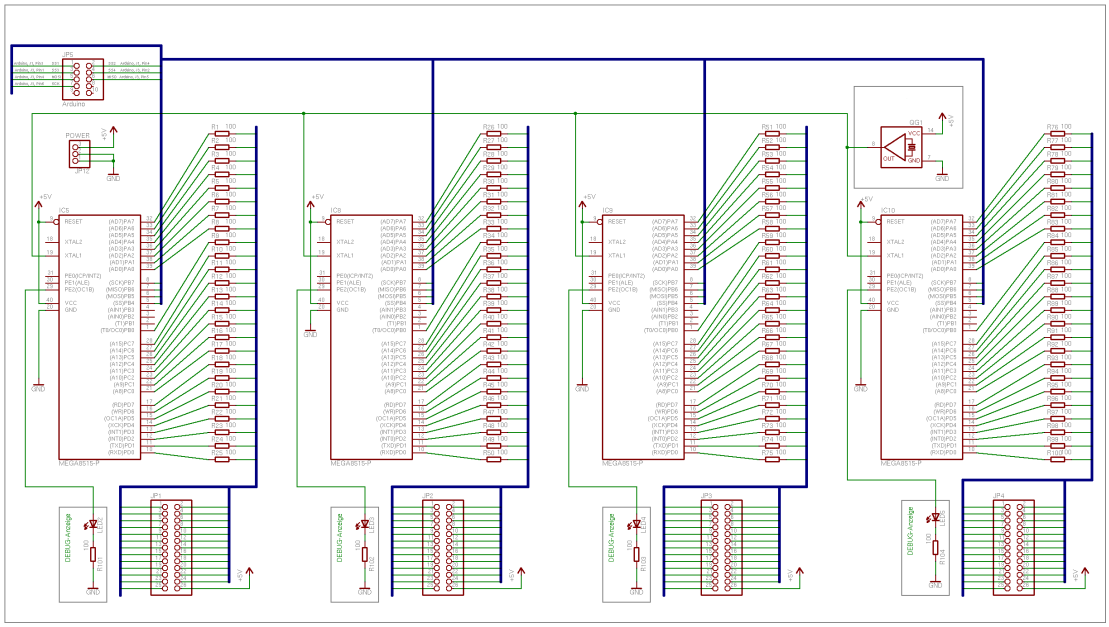

Abbildung 4.2: *Schaltplan der Säulensteuerung*

4.3 Stromversorgung

In Abbildung 4.3 ist der Schaltplan der Stromversorgung zu sehen. Der Schaltplan orientiert sich weitesgehend an dem Schaltplan aus dem Datenblatt [?] des Spannungsreglers. Das Netzteil wandelt dabei mit Hilfe eines intern Transformators die 220 Volt Eingangsspannung in 9 bis 12 Volt Ausgangspannung um. Die Ausgangsspannung wird dabei durch den Gleichrichter in eine pulsierende Gleichspannung umgewandelt. Um den gewünschten Versorgungswert von 5 Volt zu erreichen wird ein Spannungsregler eingesetzt. Kondensatoren auf der Stromversorgungsplatine glätten die pulsierende Gleichspannung. An den ICs wurden zusätzlich Kondensatoren angebracht um eine gewisse Restwelligkeit zu glätten und kleinere Störungen in der Spannung auszugleichen.

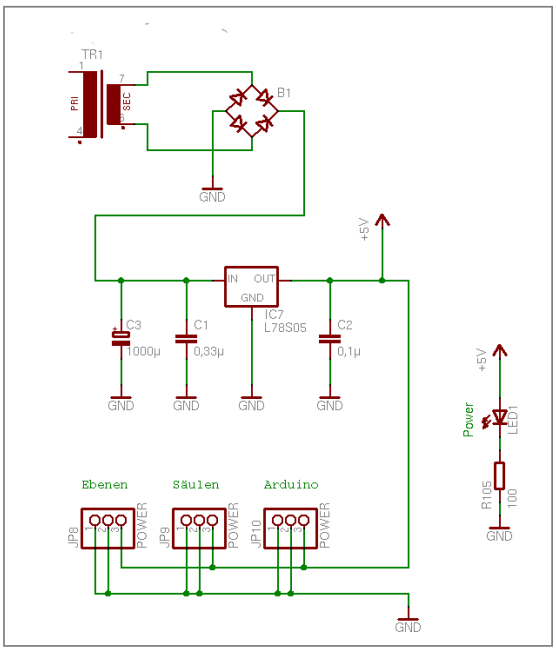

Abbildung 4.3: *Schaltplan der Stromversorgung*

5 Implementierung

Das Kapitel gibt einen Überblick über die entwickelte Hard- und Software. Wie bereits erwähnt ist der Arduino das Herz des Lichtwürfels. Von dem Arduino aus werden die Schaltvorgänge für die Säulen und Ebenen vorgenommen. Die Stromversorgung versorgt die Säulen- und Ebenensteuerung sowie den Würfel mit Strom.

5.1 Arduino

Der Arduino ist das Herz der Lichtwürfel Architektur. Von dem Arduino aus werden die Säulendaten per SPI an die Säulenplatine verschickt und die Ebenen umgeschaltet. Zur Ebenenumschaltung wird eine Interrupt Service Routine genutzt, die aufgerufen wird wenn der Interrupt Timer Overflow ausgelöst wird. In der Interrupt Routine werden zum einen per SPI die Daten zur Ebenen- umschaltung verschickt und zum anderen die vier Pins für den Decoder gesetzt mit dem die Ebenen geschaltet werden. Der Arduino ist über die USB Schnittstelle mit dem Steuerrechner verbunden, der Säulendaten an den Arduino schickt. Für die Steckre zur Ebenen- und Säulenplatine musst ein anderer Stecker auf den Arduino gelötet werden.

5.1.1 Belegung

Die I/O Pin Belegung des Arduino unterscheidet sich von der Pin Belegung die man aus dem Schaltplan entnehmen kann. Auf der Platine bedruckt werden die Pins von 0 an hoch gezählt, im Schaltplan ab 1 - wie man auch in der Tabelle sehen kann.

Arduino	Schaltplan	Funktion
0	J1/1	Serieller Bus: Rx
1	J1/2	Serieller Bus: Tx
2	J1/3	SPI: Slave Select 1
3	J1/4	SPI: Slave Select 2
4	J1/5	Ebene 0
5	J1/6	Ebene 1
6	J1/7	Ebene 2
7	J1/8	Ebene 3
8	J3/1	SPI: Slave Select 3
9	J3/2	SPI: Slave Select 4
11	J3/4	SPI: Master Output Slave Input
12	J3/5	SPI: Master Input Slave Output
13	J3/6	SPI: Serial Clock

5.1.2 Software

Programme in der Arduino Programming Language besitzen immer zwei Routinen und zwar setup() und loop(). In der setup() Routine werden Register und Variablen initialisiert, während die loop() Routine während der gesamten Lebenszeit des Programms immer wieder ausgeführt wird, also solange wie der Arduino mit Strom versorgt wird. In der Initialisierungsroutine werden die IO Pins des Arduino entsprechend der Funktionalität auf Eingang oder Ausgang geschaltet, die Baudrate für die serielle Kommunikation mit dem Steuerrechner festgelegt, der Timer Overflow Interrupt aktiviert und die Prescaler für die SPI Kommunikation und den Timer Overflow eingestellt. Für den Timer Overflow Interrupt wurde eine Interrupt Service Routine definiert. In der ISR werden die Ebenen zum einen per SPI umgeschaltet, das bedeutet die Säulen erhalten ein Paket das sie darüber informiert das sie eine Ebene umschalten müssen und welche Ebene, und zum anderen die Ausgänge des Decoders je nach Ebene auf High oder Low gesetzt. Die Codierung für die Ebenen wurde aus dem Handbuch des Decoder Bausteins übernomme. In der Hauptroutine die periodisch immer wieder aufgerufen wird, werden neue Säulendaten von der seriellen Schnittstelle gelesen und per SPI an die betreffende Säule geschickt. Der ausführlich dokumentierte Quelltext des Arduino Programm befindet sich im src/sketchbook Verzeichnis des Projektverzeichnisses.

5.2 Steuerung der Säulen

Auf der Säulensteuerung befindet sich ein externe Oszillator der mit 16 Mhz den internen Crystal der ATmega8515 anregt. Normalerweise läuft ein ATmega8515 mit einem interne Takt von 1 Mhz, was allerdings zu wenig für die SPI Kommunikation ist. Auf der Säulensteuerung sind 4 Stecker angebracht an denen jeweils 25 Säulen angeschlossen sind. Die Platine wird mit einem 6poligen Kabel mit dem Arduino verbunden. Das Kabel führt die 4 Slave Select Leitung sowie den Master Output Slave Input und die Serial Clock auf die Säulenplatine.

5.2.1 Belegung

In der folgenden Tabell ist die Pin Belegung der Säulen zu entnehmen. Die Belegung ist auf allen vier ATmegas identisch.

Pin	Funktion	Anmerkung
1	LED 9	
5	SCK	Serial Clock
6	MISO	Master Input Slave Output
7	MOSI	Master Output Slave Input
8	SS	Slave Select
9	RST	Reset
10 bis 17	LED 25 - 18	
19	XTAL 1	Externe Taktquelle
20	GND	Masse
21 bis 28	LED 17 - 10	
29	Debug LED	
32 bis 39	LED 8 - 1	

5.2.2 Software

Zu Beginn des Programms werden unterschiedliche defines für die SPI Kommunikation gesetzt, die Werte entsprechen den Pins auf dem ATmega. Der Buffer der die Säulendaten hält wird in mit einer Größe von 250 Byte initialisiert. Die Initialisierungsfunktion in initialisiert die IO Pins. Mit Hilfe der Data Direction Register (DDR) werden die Pins als Ausgänge (logisch 1) bzw. Eingänge (logisch 0) definiert. Die IO Pins auf einem ATmega sind in Ports unterteilt, daher existiert pro Port ein DDR für den Port. In einer Schleife werden die Werte im Buffer abwechselnd mit 0 (Zustand ein) und 1 (Zustand

aus) initialisiert. Die Funktion receiveData() empfängt Daten vom SPI Bus, dabei wird solange gewartet bis im Status Register (SPSR) das Interrupt Flag (SPIF) 1 ist und damit signalisiert wurde das die Übertragung beendet wurde. Das Ergebnis steht im Daten Register (SPDR) (Zeile). Die Funktion switchRow() schaltet die entsprechende Ebene um. Die für die Säule gültigen Werte werden aus dem Buffer ermittelt. Die korrekte Position wird über einen Index der mit betroffenen Ebene multipliziert wird. Sollen die Werte beispielsweise für die Ebene 4 geschaltet werden, so stehen die Werte für Ebene 4 ab der Stelle 75 im Buffer und werden entsprechend auf den Port ausgegeben. In der main Funktion werden die Daten in einer Schleife vom SPI Bus entgegen genommen. Handelt es sich beim ersten Byte um ein 0xEE so ist als Folgebyte eine Ebene zu erwarte. Ist das erste Byte ein 0xFF so sind 250 Byte Säulendaten zu erwarten die im Buffer zu aktualsieren sind. Der ausführlich dokumentierte Quelltext der Säulensteuerung befindet sich im src/avr/spi Verzeichnis des Projektverzeichnisses.

6 Bewertung

Der Zeitaufwand für das Projekt wurde stark unterschätzt. In der Planung war der April für Lötarbeiten eingeplant und der Mai für den Aufbau, so dass im Juni Zeit genug für die Programmierung bleiben würde um im Juli noch einige Feinarbeiten vor der Abgabe durchzuführen. Die Aufbauarbeiten haben sich bis in den Mai erstreckt und in der Hardware selbst gab es noch einige Fehler zu beheben.

Das Projekt selbst ermöglichte es sich näher mit der Programmierung der Atmel Prozessoren zu beschäftigen und im besonderen der Fehlerdiagnose. Da es ohne weiteres nicht möglich mit einem Debugger den Programmverlauf zu verfolgen und Inhalte von Variablen und Registern zu beobachten, muss man andere kreative Wege finden den Fehlern auf die Spur zu kommen. Kreative Wege galt es auch im Auf- und Zusammenbau des Würfels zu beschreiten. Nachdem das fräsen von Nuten in Holzplatten als Vorlage für die LED Ebenen aus Kostengründen entfiel galt es eine günstigen Weg zu finden. Am meisten Kopfzerbrechen hat sicherlich die Verdrahtung bereitet, die aber mit Silberdraht als Material die beste Lösung darzustellen scheint.

Der Arbeitsaufwand für ist relativ hoch und so würde sich für eine zukünftige Vergabe des Projektes es sich empfehlen eine Gruppengrösse von 2 bis 3 Personen anzustreben. Mit mehr Teilnehmern lässt sich das löten schneller erledigen und die Aufbauzeiten verkürzen und zum anderen könnten ein oder zwei Teilnehmer bereits mit dem programmieren beginnen.

7 Ausblick

Das Projekt ist noch nicht beendet. Während der Implementierung der Software und dem Aufbau des Würfels sind Probleme aufgetreten, die immer wieder Kopfschmerzen verursacht haben und nicht auf Anhieb zu lösen waren. Das Problem der Nebenläufigkeit durch den Interrupt im Arduino konnte durch eine Art Semaphore behoben werden. Semaphore sind in der Arduino Standardbibliothek als auch in der avr-libc nicht implementiert, daher musste man auf einen einfache boolsche Variable zurückgreifen. Der erste Ansatz war den Interrupt während der Übertragung von Säulendaten zu deaktivieren, was aber den Nebeneffekt hatte das überhaupt nicht mehr Ebenen (hardwareseitig), ausreichend schnell, umgeschaltet wurden. Die Semaphore deaktiviert nun den SPI Teil im Interrupt, wenn in der Hauptroutine gerade Säulendaten per SPI übertragen werden.

Am interessantesten waren mit Sicherheit die Effekte die während der Testphase der einzelnen Änderungen auftraten. Aktuell aktiviert und deaktiviert der Würfel einzelne Säulen, was zwar sonderlich hübsch anzusehen ist, aber nicht so ganz zu erklären ist. Das verteilen der Aufgaben auf 5 MCUs hat die Projektaufgabe nicht vereinfacht und so wäre zu überlegen, wenn man das Projekt noch einmal vergeben würde, ob man nicht versucht das ganze auf eine MCU abzubilden.

Neben den funktionellen Problemen sind an der bestehenden Software sicher einige Schönheitskorrekturen durchzuführen. Die Semaphore in der aktuellen Version der Bibliothek sind nur kompatibel mit Linux. Um die Entwicklung auch auf anderen Plattformen zu erleichtern müsste man den Teil mit der Implementierung für Semaphore für andere Systeme wie BSD Unix erweitern. In einem nächsten Schritt könnte man die Bibliothek um Socketfunktionalität erweitern so dass auch der Würfel über ein lokales Netz ansteuerbar ist. Der Aufwand dafür ist überschaubar.

Weiterhin wäre es interessant die Steuerung zu ändern und einen FPGA anstatt eines Decoders und einer Platine aus vier MCU einzusetzen. Entwicklungskits für FPGAs sind bereits für 160 Euro erhältlich. Der Einarbeitungsaufwand ist wahrscheinlich niedrig,

da die Grundkenntnisse für VHDL bereits aus der Lehrveranstaltung Digitaltechnik im Grundstudium bekannt sind.

Abbildungsverzeichnis

Literaturverzeichnis

[Atm03] Atmel. *ATmega8515 Manual*, 2003.

[SGS93] SGS-Thomson. *HC4514: 4 TO 16 LINE DECODER/LATCH Manual*, 1993.

[STS06] STS-Thomson. *L78S00 Series Manual*, 2006.