Technische Informatik 2

Projekt: Radarbrille

Gruppe: Ultraschall

Alexander Borisov,
Benny Schwarz,
Moritz W. Lemm,
Dorian Weber

12. Juli 2006

Humboldt Universität zu Berlin
Institut für Informatik

Inhaltsverzeichnis

1 Vorwort

Geneigter Leser,
wir freuen uns, Ihnen das Ergebnis unserer Arbeit präsentieren zu können. Nach vielen Stunden und langen Nächten, sind wir froh, dass Großmutter endlich angekommen ist und hoffen, dass Sie beim Lesen genauso viel Freude haben, wie wir beim Entwickeln des Projektes hatten.
Auf den nun folgenden Seiten stellen wir Ihnen eine Brille vor, die Sie sich auf die Nase setzen können. Natürlich ist das nichts spektakuläres, deshalb haben wir uns gedacht, wir könnten zusätzlich ein paar nützliche Funktionen einbauen. Das dabei entstandene Produkt sieht nicht nur modisch aus, sondern bietet zusätzlich die Möglichkeit Entfernungen und Geschwindigkeiten messen zu können. Es wurde darauf geachtet, dass der Energieverbrauch möglichst gering ist und die Umwelt nicht durch den laufenden Betrieb belastet wird.
Nun wünschen wir Ihnen viel Spaß beim Lesen.

Team Ultraschall
≪*Ultraschall bringt Bewegung in die Sache*≫

2 Darstellung der Problemstellung

2.1 Grundidee

Unsere Idee ist es, eine Brille zu entwickeln, die genau wie eine gewöhnliche Brille aussieht, aber mit der man zusätzlich Abstand und Geschwindigkeit von in Blickrichtung befindlichen Gegenständen messen kann. Die Ergebnisse werden im Blickfeld auf der Innenseite des getönten Glases ausgegeben und ständig aktualisiert. Wie der Name unserer Gruppe schon andeutet, benutzen wir dazu Ultraschall und gewisse physikalische Effekte (Dopplereffekt). Da ein Brillenglas für gewöhnlich näher als die 25 cm deutliche Sehweite vom Träger entfernt ist (man kann es mit dem Auge nicht fokussieren), konstruieren wir ein System von Spiegeln, um den benötigten Lichtweg zu erreichen.

2.2 Problemanalyse

Um Entfernungen zu messen, ohne selbst mit einem Maßband loszulaufen, müssen wir entweder jemand anderes überzeugen, diese Aufgabe für uns zu übernehmen, oder ein Signal schicken, dass von alleine zurück kommt. Wie Fledermäuse wollen wir den Schall dazu benutzen, da er recht leicht beherrschbar ist und wir auf Blackboxen verzichten können (die Funktionsweise eines Mikrophons ist uns bekannt).
Als Sensor bietet sich da natürlich das Richtmikrophon als Sender und Empfänger an, welches ständig (pro Takt) zwischen Senden und Empfangen umgeschaltet wird.

Wir benutzen das Weg-Zeit-Gesetz und den Dopplereffekt, um Entfernungen und Geschwindigkeiten eines Objektes zu bestimmen. Grundsätzlich gehen wir davon aus, dass sich der Träger der Brille auf Meeresniveau befindet, der Luftdruck konstant und die Temperatur ungefähr 20°C ist, damit wir die Schallgeschwindigkeit als Konstante betrachten können.
Prinzipiell wird der Schall vom Sensor losgeschickt, reflektiert an einem fernen Objekt und kommt wieder zurück. Die Zeitdifferenz ist messbar und daraus lassen sich mit den oben genannten Ansätzen auch die gesuchten Größen berechnen. Um die größtmögliche Genauigkeit zu erzielen, messen wir die Zeit in Takten, da das die bestmögliche Auflösung der Zeit bietet.
Daraus resultieren dann folgende physikalische Betrachtungen:

für die Entfernung

$$s = V * t$$

$$s = V * \frac{Takte}{2 * Taktfrequenz}$$

$$s = 338m/s * \frac{Takte}{2MHz}$$

$$s = 169cm * \frac{Takte}{10000}$$

für die Geschwindigkeit

Der Dopplereffekt der Akustik ist ein alltägliches Phänomen, das wir jetzt ausnutzen. Wenn sich eine Schallquelle auf einen Beobachter zu oder sich von diesem weg bewegt, ändert sich die Frequenz des Schalls. Die Höhe des Tones, die davon abhängt, wird im ersten Falle höher, da die Frequenz sich erhöht und im letzten Falle tiefer, da die Frequenz sinkt.
Dieser Effekt ist in Abbildung 1 illustriert.

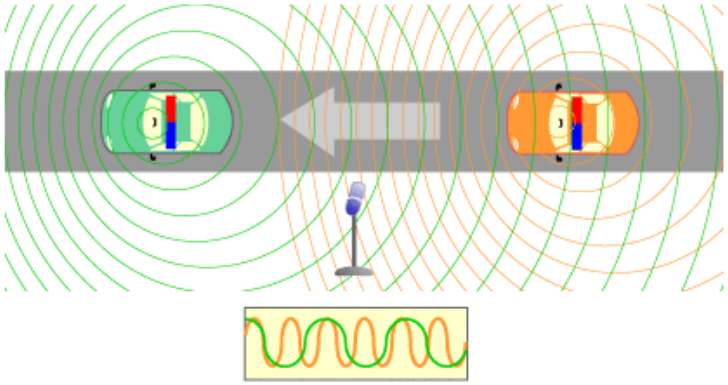

Abbildung 1: akustischer Dopplereffekt [1]

Die Brille selbst benutzt nicht den Dopplereffekt der Schallfrequenz, sondern den der Pulsfrequenz. Das Prinzip ist allerdings das Gleiche.
Mittels des Mikrophons werden 16 unterscheidbare Signale in einem bestimmten, festen Zeitabstand gesendet und die ankommende Pulsfrequenz ausgewertet. Ist diese größer als die feste Ausgangs-Pulsfrequenz, wissen wir, dass sich das anvisierte Objekt auf uns zu bewegt.
Mittels folgender Betrachtungen, lässt sich die genaue Geschwindigkeit relativ zum Träger der Brille bestimmen:

$$V = \frac{s}{t}$$

$$V_{Objekt} = \Delta s * Pulsfreq$$

$$\Delta s = V_{Schall} * (t1 - t2) * \frac{1}{2}$$

$$(t1 - t2) = \frac{(Takte1 - Takte2)}{Taktfreq}$$

$$V_{Objekt} = V_{Schall} * \frac{(Takte1 - Takte2)}{2 * Taktfreq} * Pulsfreq$$

$$V_{Objekt} = 338m/s * \frac{(Takte1 - Takte2)}{2MHz} * 10000Hz$$

$$V_{Objekt} = 338m/s * \frac{(Takte1 - Takte2)}{2000000Hz} * 10000Hz$$

$$V_{Objekt} = 338m/s * \frac{(Takte1 - Takte2)}{200}$$

$$V_{Objekt} = 152km/h * \frac{(Takte1 - Takte2)}{25}$$

Die Zeit in Takten muss immer durch 2 geteilt werden, weil der Schall den Weg hin und zurück läuft, uns aber nur die ein-fache Entfernung interessiert.

Ausgabe im Sichtfeld

Bleibt noch das Problem mit der Ausgabe zu klären. Als normalsichtiger Mensch kann man Dinge im Nahbereich unter 25 cm nicht mehr scharf fokussieren, was an der Funktionsweise unseres Auges liegt. Da unsere Datenausgabe möglichst gut sichtbar sein sollte, möchten wir sie auf das Brillenglas ausgeben, das offensichtlich weniger als die deutliche Sehweite vom Auge entfernt ist.
Dazu konstruieren wir ein Spiegellabyrinth gemäß der Abbildung 4, um den nötigen Lichtweg zu erreichen. Über eine starre Bügelkonstruktion wird die digitale Zahl auf das Glas gebracht. Abbildung 5 zeigt wie.

2.3 Basisfunktionalitäten und Systemanforderungen

Features

- Distanzmessung bis 1 km Entfernung, mit einer Genauigkeit von 4 cm
- Geschwindigkeitsmessung bis 200 km/h auf 1 km/h genau
- Ausgabe ins Sichtfeld, 2 mal pro Sekunde aktualisiert
- Geringer Energieverbrauch, ungekühlt einsetzbar

Daraus resultierende Systemanforderungen

- Getrennte Taktgeber für Sender und CPU
- Sender möglichst hoch getaktet, für gute Genauigkeit in der Zeitmessung (taktet mit 1 MHz)
- CPU niedrig getaktet, um möglichst wenig Leerlauf zu haben (342 Hz)
- Busbreite ergibt sich aus der Reichweite der Messwerte (24 Bit breit)
- Asynchrone Verarbeitung der CPU relativ zum Sender (verschiedene Taktgeber)

2.4 Schaubilder

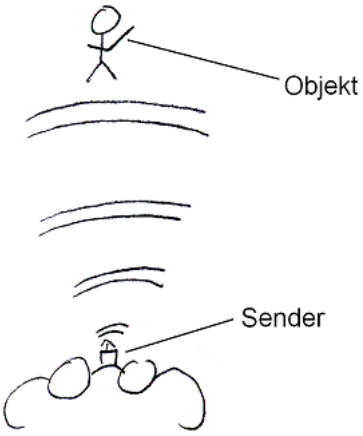

Abbildung 2: Senden

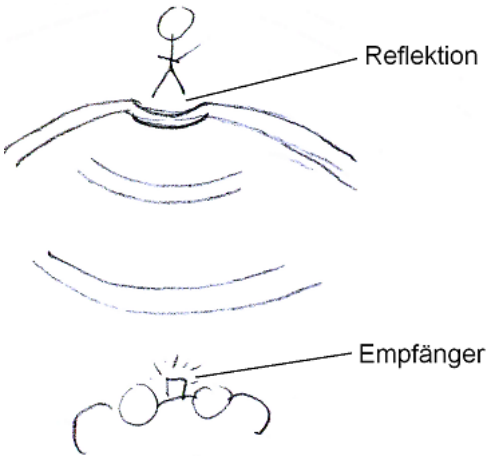

Abbildung 3: Empfangen

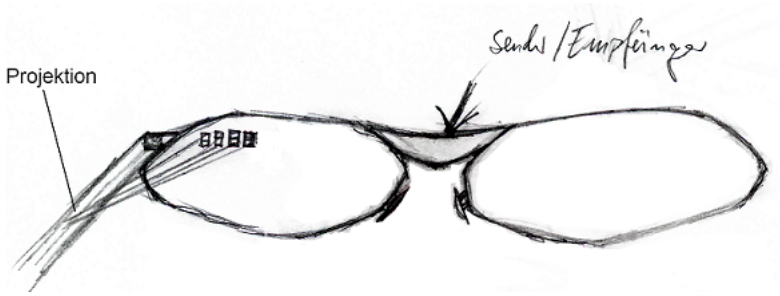

Abbildung 4: Design

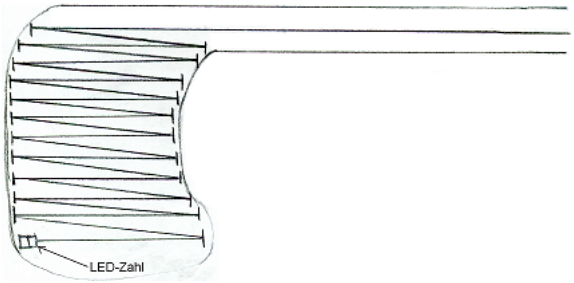

Abbildung 5: Verlängerung des Lichtweges

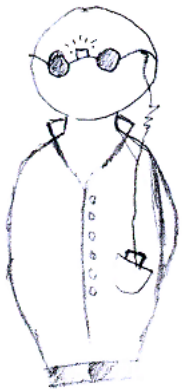

Abbildung 6: Im Einsatz

3 Architekturentwurf (logische Ebene)

3.1 Datengewinnung, Datentypen und -formate

Das Mikrophon sendet kontinuierlich Signale. Um diese Signale beim Eingang zuordnen und unterscheiden zu können, wird die Signalfrequenz in Abhängigkeit vom Timer immer um 500Hz inkrementiert, startend bei 36kHz (Ultraschallbereich). Die eingehenden Signale können dann anhand ihrer Frequenz zugeordnet werden. Das ist notwendig, um den Offset des Timers gegenzurechen (wir wollen wissen, wie lange das Signal unterwegs war und nicht zu welcher absoluten Zeit es angekommen ist) und um die gemessene Zeit an der entsprechenden Stelle in den Speicher zu schreiben. Dem Mikrophon wird von einem Zähler, der in Abhängigkeit vom Timer zählt, eine ganze 4 Bit Binärzahl zwischen 0 und 15 übergeben. Das Mikrophon wandelt diese Zahl in eine Frequenz um und sendet diese. Analog werden eingehende Signale vom Mikrophon verifiziert (Toleranz 100Hz) und in eine ganze Binärzahl zwischen 0 und 15 umgewandelt. Diese Zahl ist unsere Adresse im RAM, in welche die Zeit geschrieben wird. Im Speicher stehen zu jeder Zeit 16 ganzzahlige Werte, die mit jedem Zyklus des Timers aktualisiert werden und nach Belieben von der CPU abgegriffen werden können um dort weiterverarbeitet zu werden.

Da wir aufgrund der langen Laufwege des Schalls damit rechen müssen, dass immer mal wieder einzelne Signale verloren gehen, bleiben prinzipiell alle gemessenen Zeiten solange im Speicher stehen, bis sie in einem späteren Zyklus überschrieben werden. Die Auswertung ist dadurch zwar nicht zu jeder Ausgabe korrekt, konvergiert aber zum richtigen Wert. Ist eine Messung aussagekräftig, dann pegeln sich die Werte ein und die Ausgabe zeigt dauerhaft dieselben Werte. Ist die Messung noch nicht abgeschlossen, dann verändern sich die Werte der Ausgabe und der Benutzer weiß, dass er noch keine aussagekräftige Messung hat.

3.2 Verarbeitung und Speicherung

Da die gemessenen Frequenzen dem akustischen Dopplereffekt unterliegen, müssen sie mit einer Toleranzgrenze von 100Hz gerundet werden. Bei Frequenzen, die nicht zugeordnet werden können, wie z.B. Lärm in der Messumgebung, werden so die entsprechenden nicht in den RAM übernommen.

Sind 16 Werte mittels Snapshot (SNP) in den Stack der CPU übernommen, wird das arithmetische Mittel errechnet (siehe Assembler Code) sowie die mittlere Differenz zwischen den einzelnen Zeiten. Die Zeiten und Differenzen werden als ganzzahlige Binärzahlen mit Vorzeichen im Zweierkomplement gespeichert. Mit der gemittelten Zeit wird die Entfernung, mit der mittleren Differenz die Geschwindigkeit des betrachteten Objekts errechnet. Ist die CPU mit der Rechnung fertig, werden durch SNP zusätzlich die errechneten Daten ausgegeben und die Neuen aus dem RAM in den Stack geholt.

3.3 Datenausgabe

Die auszugebenen Daten werden den LED-Blöcken als Binärwerte übergeben, von den LEDs angezeigt und über das Spiegellabyrinth auf kleine Spiegel in den

Innenseiten der Brillengläser projeziert. Die Anzeige der Entfernung enthält an der zweiten Stelle von rechts ein festes Komma.

4 Architekturentwurf (technische Ebene)

4.1 Vorbemerkungen

Wie wir bereits wissen, besteht ein Rechner im Allgemeinen aus drei Grundkomponenten:

- Prozessor (Central Processing Unit: CPU)
- Speicher
- Ein-/Ausgabe-Einheit (I/O)

Dazu kommen natürlich noch Verbindungen (Busse) zwischen den Einheiten. Dabei übernimmt die CPU die Ausführung der Befehle, sowie die Ablaufsteuerung. Im Speicher werden Daten und Programme binär kodiert abgelegt. Über die I/O–Einheit kann man die Daten eingeben bzw. einlesen, und ausgeben. Abbildung 7 zeigt den prinzipiellen Aufbau von unserem Rechner.

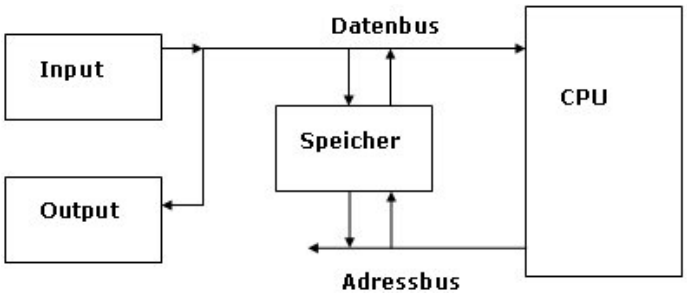

Abbildung 7: Allgemeine Struktur

4.2 Aufbau

In diesem Abschnitt wird nun genau erklärt, wie der Prozessor in unserem Projekt aufgebaut ist. Unsere CPU besteht aus zwei Teilen: aus einem Datenprozessor und einem Befehlsprozessor.

Datenprozessor

Die Aufgabe des Datenprozessors, ist die Verarbeitung von Daten, was zum Beispiel die Berechnung von Werten oder das Ausführen von bestimmten Anweisungen sein kann. Um das realisieren zu können, enthält unser Prozessor, wie üblich, eine arithmetisch-logische Einheit (ALU), sowie Register zur Aufnahme von Operanden. Bei den Registern handelt es sich um den Akkumulator und ein "General Purpose"-Register, das die zweiten Operanden aufnehmen kann. Mögliche Flags bei Rechnungen, wie zum Beispiel die Zero-Flag, werden intern in der ALU gesichert.

Befehlsprozessor

Die Aufgabe des Befehlsprozessors besteht darin, Befehle zu entschlüsseln und deren Ausführung zu steuern. Dazu gibt es folgende Register:

- Befehlregister (IR - Instruction Register): Hier befindet sich der aktuell bearbeitete Befehl.
- Befehlzähler (PC - Program Counter): Hier wird die Adresse des nächsten auszuführenden Befehls gespeichert.
- Speicheradressregister (MAR - Memory Address Register): Hier wird die Adresse des Speicherplatzes, welcher als nächstes anzusprechen ist, abgelegt.

Darüber hinaus haben wir noch zwei Speicher. Speicher 1 befindet sich direkt in der Brille. Dort werden die 16 Messwerte gespeichert, die wir als Paket absenden. Sobald diese nicht mehr benötigt werden, werden sie überschrieben. Im Speicher 2 werden dann die eigentlichen Ergebnisse festgehalten, bzw. die Zwischenoperationen abgelegt.

Die Abarbeitung der Befehle sieht folgendermaßen aus:

1. Interpretationsphase: Die Adresse aus dem PC wird über den Adressbus aus dem EEPROM Programmspeicher geholt und über den Befehlsbus in das IR geladen. Die CPU geht davon aus, dass diese Bitfolge ein Befehl ist. Der Dekodierer erkennt dann, um welchen Befehl und Befehlstyp es sich handelt.
2. Ausführungsphase: Hier erfolgt die eigentliche Befehlausführung, sowie die Initiierung der Interpretationsphase für den nächsten auszuführenden Befehl.

4.3 Erläuterung der Komponenten

Datenbus

Breite von 24 Bit - Speicherung von Zahlen mit der Reichweite 2^{24} im Zweierkomplement.

Adressbus

Breite von 6 Bit - Zugriff auf maximal 64 Speicherstellen.

CPU/Clock

Getaktet auf 342 Hz, 3 Takte pro Befehl, eine halbe Sekunde pro Programmdurchlauf.

CPU/A

Akkumulatorregister. Typischer Bestandteil einer 1-Adress-Registermaschine.

CPU/R

General Purpose Register.

CPU/ALU

Arithmetisch-logische Einheit. Dient der Berechnung von Entfernung/Geschwindigkeit, Bildung von Mittelwerten, Zahlenvergleich.

CPU/IR

Instruktionsregister. Enthält den aktuell zu bearbeitenden Befehl.

CPU/PC

Programmzähler. Speicherung der Adresse des nächsten auszuführenden Befehls.

CPU/MAR

Memory-Adress-Register. Adresse des als nächstes angesprochen Speicherplatzes.

Speicher 1 und CPU/Speicher 2

Enthält Platz für 16 Maschinenwörter, die jeweils 3 Bytes lang sind. Dieser wird für Mess- und Zwischenergebnisse benötigt.

Sender/Empfänger

Richtmikrofon. Ermöglicht das Senden und Empfangen von Ultraschallwellen.

Ausgabe

LED-Anzeige. Ermöglicht die Anzeige der errechneten Messwerte.

4.4 Schaubild

Abbildung 8 stellt den genauen Aufbau unseres Prozessors dar.

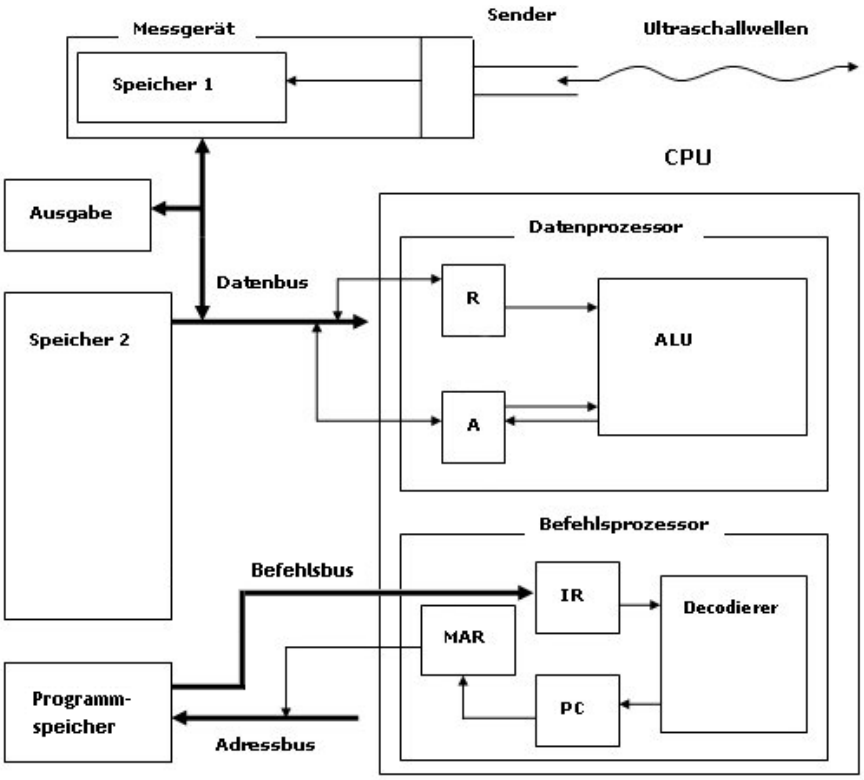

Abbildung 8: Detaillierte Struktur

5 Mikroprozessorentwurf

5.1 Spezifikation

- 16 Befehle
- Programmgröße maximal 64 Wörter
- Speicher wortadressierbar
- 16 Bit Werte im Code

Daraus ergibt sich folgendes Format:

OP	Register	Adresse	Idle
⟷	⟷	⟷	⟷
4 Bit	1 Bit	6 Bit	5 Bit

Die 5 Idle-Bits sind für Erweiterungen an der Infrastruktur verfügbar. Das gibt dem Gerät zusätzliche Flexibilität.

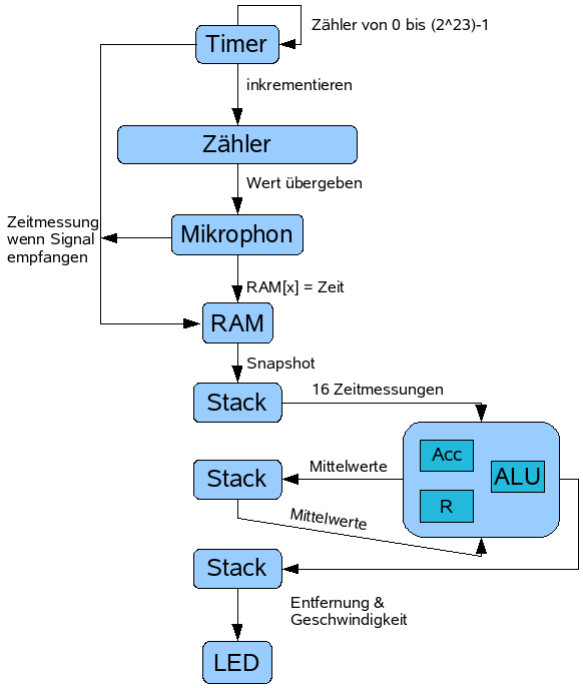

Abbildung 9: Verhaltensmodell

5.2 Befehlssatz

Opcode	Mnemonic	Aktion
0000_2	NOP	No-Operation
0001_2	CALL Addr	$sp = 0$, $Acc = 0$, $R = 0$
0010_2	SNP	kopiert alle Daten von SP1 zu SP2
0011_2	HALT	schaltet das Gerät aus
0100_2	ADD R_x	$Acc = Acc + R_x$
0101_2	SUB R_x	$Acc = Acc - R_x$
0110_2	MUL R_x	$Acc = Acc * R_x$
0111_2	DIV R_x	$Acc = Acc/R_x$
1000_2	JMP Addr	$PC = Addr$
1001_2	JEQ Addr	Zero-Flag $= 1 \Rightarrow PC = Addr$
1010_2	JNP Addr	Pop-Flag $= 0 \Rightarrow PC = Addr$
1011_2	JGR Addr	Greater-Flag $= 1 \Rightarrow PC = Addr$
1100_2	POP R_x	$R_x = \text{SP2}[++sp]$
1101_2	MOV R_x	$R_x = (++PC)$
1110_2	PEEK R_x	$R_x = \text{SP2}[sp]$
1111_2	POKE R_x	$\text{SP2}[sp] = R_x$

5.3 Steuereinheit

5.3.1 Mikrocode

Zum Steuern unseres Prozessors benötigen wir insgesamt 25 Steuerbits. Eine Übersicht der Bits findet sich in der folgenden Tabelle. Zur Adressierung im Mikroprogrammspeicher benötigen wir 6 zusätzliche Bits, um die folgende Mikroinstruktion zu bestimmen.

Adresse	Register	EEPROM	Stack	ALU	Weitere Bits
$\longleftrightarrow$	$\longleftrightarrow$	$\longleftrightarrow$	$\longleftrightarrow$	$\longleftrightarrow$	$\longleftrightarrow$
6 Bit	10 Bit	3 Bit	5 Bit	4 Bit	3 Bit

Tabelle 1: ein Mikrowort

1. Register Bits
   R0_IN = speichere Datenbuswert in Register R0
   R0_OUT = lege Wert in Register R0 auf Datenbus
   ACC_IN = speichere Datenbuswert in Register R0
   ACC_OUT = lege Wert in Akkumulator auf Datenbus
   ALU_IN = gebe Wert im Akkumulator für ALU frei
   ALU_OUT = gebe Wert aus ALU zur Speicherung im Akkumulator frei
   PC_IN = gebe den Adress-Wert aus IR für den PC frei
   PC_OUT = gebe den Adress-Wert aus PC für MAR frei
   INC_PC = Steuerbit zum Inkrementieren des PC
   IR_IN = lade Befehl aus EEPROM ins IR

2. EEPROM Befehlsspeicher Bits
   RR (Read from ROM) = Steuerbit für Lese-Zugriff auf ROM

WFR (Wait For ROM Fetch Complete) = Bit zur Steuerung des EEPROM Zugriffs
PUT_ROM = gibt Wert aus EEPROM für Datenbus frei

3. Stackspeicher Bits
   RS (Read from Stack) = Steuerbit für Lese-Zugriff auf Stack-Speicher
   WS (Write To Stack) = Steuerbit für Schreib-Zugriff auf Stack-Speicher
   WFS (Wait For Stack Fetch Complete) = Bit zur Steuerung des Stack Zugriffs
   OUTPUT = Ausgabe des Ergebnisses an Speicherstelle SP2[0]
   INC_SP = Steuerbit zum Inkrementieren des Stack-Pointers

4. ALU Bits
   ADD/SUB/MUL/DIV = Steuerbits für die ALU-Funktionen

5. Zusätzliche Bits
   Clear = ermöglicht das Löschen von R0, Akkumulator und Stackpointer
   FP (Fetch Package) = steuert die Übergabe der Werte aus Speicher 1 nach Speicher 2
   PLA (Programmable Logic Array = Decode Instruction) = Steuerung des PLA Dekoders

6. Adress-Bits
   6 Bit breite nächste Adresse

Die Bestimmung der nächsten Adresse erfolgt folgendermaßen:
Wird ein neuer Befehl aus dem EEPROM gelesen, so wird durch das PLA die Startadresse des zugehörigen Mikroprogramms bestimmt. Wird gerade ein Mikroprogramm abgearbeitet, so liegt die nächste Adresse im 6 Bit Adressfeld der Mikroinstruktion. Eine Auswahl zwischen diesen zwei Möglichkeiten erfolgt durch ein vorgeschaltetes OR-Gatter.

Zur Anschaulichkeit haben wir unser Mikroprogramm in mehrere Unterprogramme unterteilt. Zur Steuerung der Befehlausführung verwenden wir die folgenden Unterprogramme:

1. Fetch = Holen des nächsten Befehls
2. Nop = keine Operation
3. Call = implementiert den Befehl CALL Adr
4. Snap = Übernahme der Speicherdaten & Ausgabe der Ergebnisse
5. Add, Sub, Mul, Div = implementieren die Befehle ADD, SUB, MUL, DIV
6. Jump = implementiert die Sprungbefehle JMP/JEQ/JNP/JGR
7. Pop = implementiert den Befehl POP R_x
8. Move = implementiert den Befehl MOV R_x
9. Peek = implementiert den Befehl PEEK R_x
10. Poke = implementiert den Befehl POKE R_x

Mikroprogramme

```
>>Name: Fetch
>>Ablauf:
: 1. PC_OUT, RR
: 2. WFR, INC_PC
: 3. IR_IN
: 4. PLA

>>Name: Nop
>>Ablauf:
: kein Steuerbit zu setzen

>>Name: Call
>>Ablauf:
: 1. Clear, PC_IN

>>Name: Snap
>>Ablauf:
: 1. Clear
: 2. RS
: 3. WFS
: 4. OUTPUT, INC_SP
: 5. RS
: 6. WFS
: 7. OUTPUT
: 8. FP

>>Name: Add
>>Ablauf:
: 1. R0_out, ALU_IN, ADD
: 2. ALU_OUT

>>Name: Sub
>>Ablauf:
: 1. R0_out, ALU_IN, SUB
: 2. ALU_OUT

>>Name: Mul
>>Ablauf:
: 1. R0_out, ALU_IN, MUL
: 2. ALU_OUT

>>Name: Div
>>Ablauf:
: 1. R0_out, ALU_IN, DIV
: 2. ALU_OUT
```

```
>>Name: Jump
>>Ablauf:
: 1. PC_IN

>>Name: Pop
>>Ablauf:
: 1. INC_SP
: 2. RS
: 3. WFS
: 4. R0_IN

>>Name: Move
>>Ablauf:
: 1. PC_OUT, RR
: 2. WFR, INC_PC
: 3. PUT_BUS, R0_IN

>>Name: Peek
>>Ablauf:
: 1. RS
: 2. WFS
: 3. R0_IN

>>Name: Poke
>>Ablauf:
: 1. R0_OUT, WS
: 2. WFS
```

Insgesamt benötigen wir 35 Mikroinstruktionen. Der Mikroprogrammspeicher ist daher 35 x 31 Bit groß.

Anmerkungen zum Programmable Logic Array

Das PLA dient als Instruktionsdekoder. Es ist mit dem Instruktionsregister und den Statusflags verbunden. Anhand des OP-Codes des geladenen Befehls und der Statusflags berechnet es die Startadresse des zugehörigen Mikroprogrammes.

Befehl	OPCODE	Flag	Mikroprogramm	Mikroadresse
NOP	0000	x x x	Nop	4
CALL	0001	x x x	Call	5
SNP	0010	x x x	Snap	6
ADD	0100	x x x	Add	14
SUB	0101	x x x	Sub	16
MUL	0110	x x x	Mul	18
DIV	0111	x x x	Div	20
JMP	1000	x x x	Jump	22
JEQ	1001	0 1 0		22
JNP	1010	1 0 0		22
JGR	1011	0 0 1		22
POP	1100	x x x	Pop	23
MOV	1101	x x x	Mov	27
PEEK	1110	x x x	Peek	30
POKE	1111	x x x	Poke	33

Tabelle 2: Logiktabelle

x = don't care

5.3.2 Schaltung

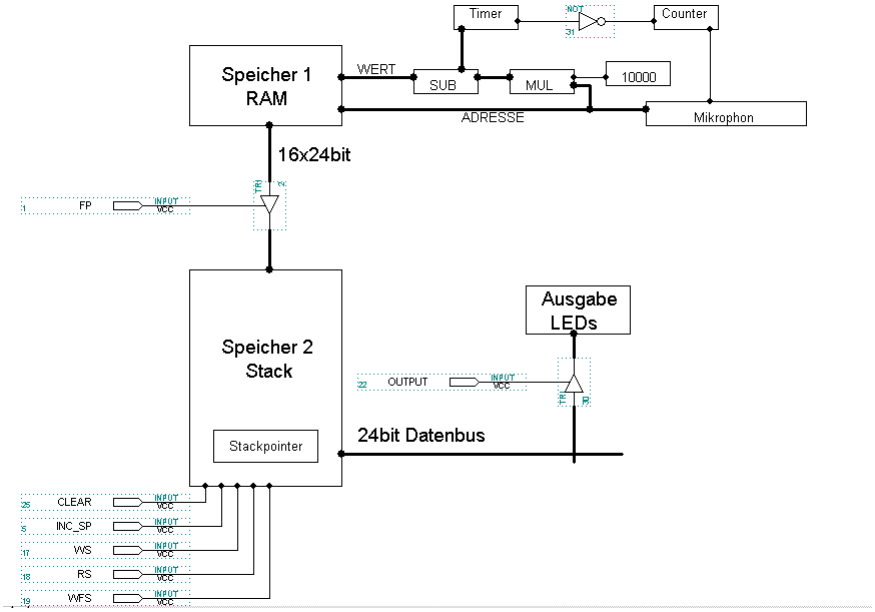

Abbildung 10: Schaltung des Brillenteils

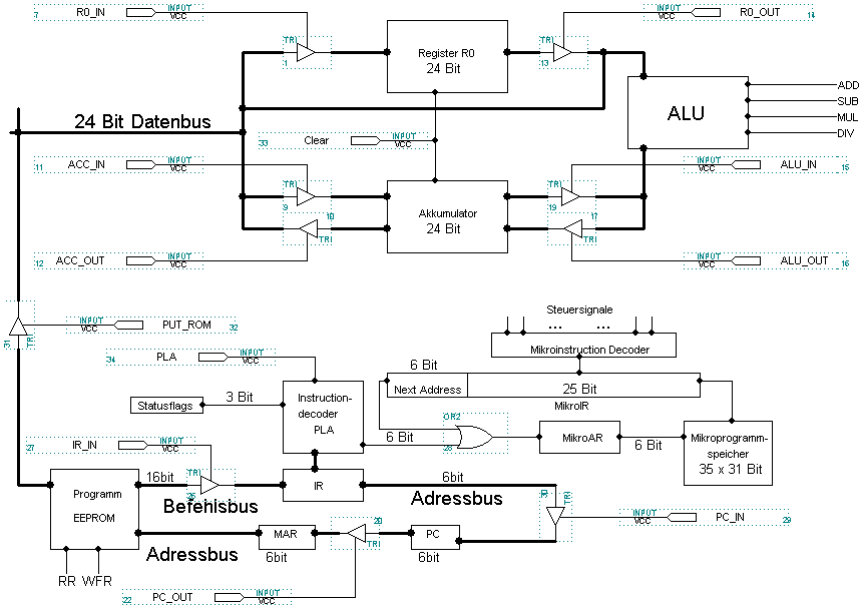

Abbildung 11: Steuerwerk

5.4 Assemblercode

5.4.1 Hauptprogramm

```
Main:     MOV #16, R
Divide:   POP Acc
          DIV R
          POKE Acc
          JNP Divide
-------------------------
          POP Acc
Mean:     POP R
          ADD R
          JNP Mean
-------------------------
Diff:     PEEK R
          POKE Acc
          POP Acc
          SUB R
          JNP Diff
          ADD R
          MOV #16, R
          DIV R
Diff_2:   POP R
          ADD R
          JNP Diff_2
          SUB R
          POP R
          POKE Acc
-------------------------
          CALL Dist
Main_2:   POKE Acc
          CALL Speed
          POKE Acc
Main_3:   SNP
          JMP Main
          HALT
```

5.4.2 Entfernung

```
Dist:    PEEK Acc
         MOV #49000, R
         SUB R
         PEEK Acc
         JGR Dist_2
         MOV #169, R
         MUL R
         MOV #10000, R
         DIV R
         JMP Dist_3
Dist_2:  MOV #10000, R
         DIV R
         MOV #169, R
         MUL R
Dist_3:  JMP Main_2
```

5.4.3 Geschwindigkeit

```
Speed:  POP Acc
        MOV #152, R
        MUL R
        MOV #25, R
        DIV R
        JMP Main_3
```

6 Schlussbemerkungen

6.1 Bewertung der geplanten Einsatzmöglichkeiten

Die Radarbrille kann vielseitig in Freizeit, Sport oder auch militärisch eingesetzt werden. So könnte sie Sondereinsatzkräften von Polizei und Militär helfen, schnell taktische Entscheidungen zu treffen.

Sportlern, die auf Sichtorientierung angewiesen sind, wie z.B. Bergsteiger, Wanderer oder Segler, wäre die Brille eine Erleichterung beim Einschätzen von Entfernungen.

6.2 Probleme

Durch die Benutzung von Schall ergeben sich recht lange Wartezeiten in denen die Messung gestört werden kann. Typisch hierfür wären z.B. fahrende Autos vor dem anvisierten Ziel, oder dass der Benutzer den Kopf dreht, bevor die Signale zurückgekommen sind.

6.3 Testmöglichkeiten

Allgemein lassen sich die Funktionen am besten durch normalen Gebrauch, also Tragen der Brille im Alltag, überprüfen. Wenn die Frequenz des Mikrophons auf eine hörbare Frequenz reduziert wird, lässt sich leicht die Fehleranfälligkeit der Brille testen (durch Erzeugen der passenden Frequenzen, z.B. durch Rufen).

Man sollte die Auswirkungen auf Tiere untersuchen, ob diese von dem für sie oftmals hörbaren Ultraschall gestört werden.
Es wurde zwar so konzipiert, dass Hunde nicht davon betroffen sind (Hunde hören bis 35kHz, das Mikrophon sendet ab 36kHz), doch es ist uns unklar, in wie weit andere Tiere darauf reagieren könnten.

6.4 Mögliche Erweiterungen

Die beste Erweiterung ist zweifelsfrei der Umstieg auf Infrarotlicht. Viele Probleme fallen dadurch weg, da sich in der Zeit zwischen zwei Messungen im Allgemeinen die Situation nicht signifikant ändern kann. Das ist deshalb gut, weil noch weniger Messwerte verloren gehen (Kopf drehen oder Bewegung von Objekten, die in den Lichtweg eintreten sind kaum möglich). Auch sehr vorteilhaft ist der Wegfall des Bedarfs nach einem Trägermedium, bei Schall Luft, womit die Brille z.B. auch in der Raumfahrt einsetzbar wäre. Dazu kommt, dass die Wartezeit zwischen zwei Messungen komplett weg fällt.
Die verwendete Technik kann fast 1:1 übernommen werden, allerdings muss in diesem Modell der Taktgeber des Sensors recht hohe Taktfrequenzen erreichen und daher unter Umständen nach dieser Methode gekühlt werden.
Wenn eine Messung mit Infrarot auf einen Meter genau sein soll, muss die Clock des Sensors eine Frequenz von $3 * 10^8$Hz haben (ein Takt pro Meter), was 300 MHz entspricht. Allerdings skaliert dann die CPU nicht mehr richtig damit, weshalb diese auch auf eine angemessene Geschwindigkeit getaktet werden sollte. Diese Komponenten müssen dann unter Umständen gekühlt werden und haben einen höheren Stromverbrauch, als unsere Lösung, die im Übrigen auf 4 cm genau misst.

7 Arbeitsaufteilung

- Benny Schwarz - Leiter für Befehlsformat und Mikrocode
- Alexander Borisov - Leiter für Architekturentwurf auf logischer und technischer Ebene
- Moritz W. Lemm - Leiter für Problemanalyse und Anforderungsprofil, Architekturentwurf auf logischer Ebene
- Dorian Weber - Leiter für Befehlsspezifikation, Assemblercode und generelle Überarbeitung
- Alle Mitglieder haben an allen Themen mitgearbeitet und sind damit vertraut.

8 Literatur

Literatur

[1] Wikipedia: Doppler-Effekt