



Projekt: Spurwechselassistent

Katharina Amboß <amboss@informatik.hu-berlin.de>
Knut Müller <kmueller@informatik.hu-berlin.de>
Kornelius Kalnbach <kalnbach@informatik.hu-berlin.de>
Enno Gröper <groeper@informatik.hu-berlin.de>
(Arbeitsgruppe: MegaTama4)

Juni/Juli 2005

Betreuer: Dr. Siegmund Sommer

Inhaltsverzeichnis

1	Problemstellung	4
1.1	Idee	4
1.2	Problemanalyse	4
1.2.1	Wann ist ein gefahrloser Spurwechsel möglich?	5
1.2.2	Welcher Bereich muss überwacht werden?	5
1.2.3	Wie kann das umgesetzt werden?	5
1.3	Basisfunktionalität	6
1.3.1	Sensorik	7
1.3.2	Prozessor	7
1.3.3	Benutzerschnittstelle	8
2	Architekturentwurf und Mikroprozessor-Umgebung	9
2.1	logische Ebene	9
2.1.1	Datengewinnung	9
2.1.2	Datentypen und -formate	10
2.1.3	Verarbeitung und Speicherung	11
2.1.4	Datenausgabe	11
2.2	technische Ebene	12
2.2.1	Schnittstellen	12
2.2.2	Sensorsteuerung	12
2.2.3	Speicherverwaltung	13
3	Mikroprozessor	13
3.1	Befehlsformat	13
3.2	Befehlssatz	14
3.3	Steuereinheit	15
3.3.1	Steuereinheit	15
3.3.2	Mikroprogrammspeicher	16
3.3.3	Dekoder	17
3.3.4	Adressgenerator	18
3.3.5	Taktunterbrecher	19
3.4	Software	21
3.5	Schaltungsentwurf	22
4	Schlussbetrachtungen	24
4.1	Einsatzmöglichkeiten	24
4.2	Einschränkungen	24

4.3	Sicherheitsbetrachtungen	25
4.4	mögliche Erweiterungen	25
5	Anhang	25
5.1	Assemblercode	25
5.2	Mikrocode	36
6	Quellen	40

1 Problemstellung

1.1 Idee

Im Alltag eines Autofahrers passiert es leider viel zu oft, dass ein Autofahrer vor einem Spurwechsel nicht ausreichend genau guckt und dadurch andere Verkehrsteilnehmer gefährdet.

Unser Ziel ist es solche Situationen zu vermeiden. Daraus haben wir die Idee für einen Spurwechselassistenten entwickelt.

Der Spurwechselassistent soll dem Fahrer beim Wechseln der Spur unterstützen, ohne ihm die Verantwortung zu nehmen. Realisiert haben wir dies, indem das System warnt, falls kein sicherer Spurwechsel möglich ist.

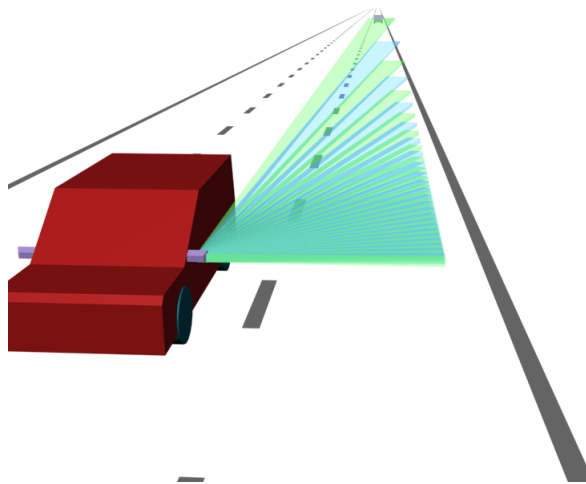


Abbildung 1: perspektivische Sicht

1.2 Problemanalyse

Wir wollen ein Fahrerassistenzsystem bauen, das gefährliche Spurwechsel verhindert. Vor einem Spurwechsel ist zu prüfen, ob der Abstand zu den Fahrzeugen in der neuen Spur ein gefahrloses Wechseln zulässt.

1.2.1 Wann ist ein gefahrloser Spurwechsel möglich?

Während des Spurwechsels darf der Sicherheitsabstand zum vorrausfahrenden bzw. zum nachfolgenden Fahrzeug nicht unterschritten werden.

Der Sicherheitsabstand definiert sich aus der Straßenverkehrsordnung nur allgemein. Die Fahrschule wird da schon genauer: Außerhalb von Ortschaften gilt der 2-Sekunden-Abstand, das entspricht der Strecke, die das Fahrzeug in zwei Sekunden zurücklegt. Bekannter ist die Regel: Die Hälfte der Tachoanzeige in Metern. Innerorts von Ortschaften gilt der 1-Sekunden-Abstand.

In unserem Fall unterscheiden wir zwischen außerorts und innerorts durch eine festgelegte Geschwindigkeit: Schneller als 60 km/h ist außerorts, langsamer als 60 km/h ist innerorts.

1.2.2 Welcher Bereich muss überwacht werden?

Es muss sichergestellt werden, dass durch den Spurwechsel kein Fahrzeug behindert wird. Ein Fahrzeug wird behindert, wenn in den Sicherheitsabstand eines Fahrzeugs auf der benachbarten Spur gefahren wird, aber auch wenn der eigene Sicherheitsabstand zu einem auf der neuen Spur voraus fahrendem Fahrzeug unterschritten wird. Um das zu verhindern, müssen die beiden benachbarten Spuren überwacht werden. Durchschnittlich ist eine Spur 3m breit, daher reicht eine Bereichsbreite von 2,8m neben dem Fahrzeug aus. Der zu beobachtende Bereich muss länger sein als der Sicherheitsabstand bei der höchstmöglichen Geschwindigkeit, da auch nach dem Spurwechsel der Sicherheitsabstand noch vorhanden sein soll. Als maximale Geschwindigkeit legen wir 300 km/h fest, da eine solche Geschwindigkeit nur von sehr wenigen Fahrzeugen zu erreichen ist. Hinzu kommt, dass die realen Straßenbedingungen es meist nicht zulassen solch hohe Geschwindigkeiten zu fahren. Damit ist eine Bereichslänge von 300m, 150m nach vorn und 150m nach hinten, ausreichend.

1.2.3 Wie kann das umgesetzt werden?

Als Eingangsdaten für den Prozessor brauchen wir Information, ob in dem überwachten Bereich ein Fahrzeug ist und mit welcher Geschwindigkeit es in welcher Entfernung zum eigenen Fahrzeug fährt.

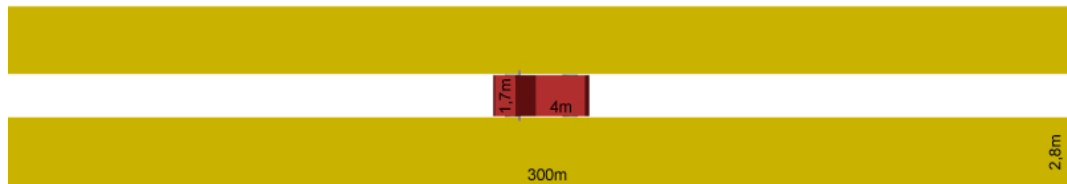


Abbildung 2: Der Bereich

Die Suche nach dem richtigen Sensor stellte sich als schwierig heraus, da der Sensor einen 300m langen und fast 3m breiten Bereich neben der Sensorposition überwachen muss. Ideal wäre eine Kamera mit nachgeschalteter Bildverarbeitung. Da dies aber sehr aufwendig ist, haben wir uns für einen Radarsensor entschieden.



Abbildung 3: Der Sensor

Da die beiden zu beobachtenden Bereiche durch das Fahrzeug getrennt sind, haben wir uns für zwei unabhängige Systeme, je eins für die beiden Fahrzeugseiten, entschieden. Das System besteht aus dem Radarsensor und dem Prozessor und kann vollständig in den Außenspiegel integriert werden.

Das System ist zusätzlich günstiger, da ein langsamerer Prozessor verwendet werden kann.

1.3 Basisfunktionalität

In diesem Abschnitt soll die Funktionsweise der einzelnen Komponenten des Spurwechselassistenten beschrieben werden.

Die einzelnen Komponenten lassen sich nach dem Prinzip Eingabe - Verarbeitung - Ausgabe einteilen: die Sensorik als Eingabe, der Prozessor zur Verarbeitung und die

Benutzerschnittstelle zur Ausgabe.

1.3.1 Sensorik

Um die Entfernung der Fahrzeuge zu messen, setzen wir einen Radarsensor ein. Der Radarsensor gibt in einem fest einstellbaren Sektor die Entfernung des nächsten Objektes wieder. Da wir auf einer Entfernung von 150m nur *eine* Spur beobachten wollen, muss der Sektor relativ klein gewählt werden. Durch Versuche haben wir festgestellt, dass eine Sektorbreite von 2° die optimale Lösung ist. Dadurch entstehen 90 Sektoren.

Um den gesamten Bereich abdecken zu können muss der Sensor gedreht werden. Die Entfernung wird dabei verfälscht, deshalb muss der Winkel der Drehung ebenfalls ermittelt werden. Der Winkelsensor sendet der Sensor-Steuereinheit alle 2° , angefangen bei 1° , ein Signal zum Übernehmen der Entfernungsdaten vom Radarsensor. Da der Sicherheitsabstand aus absoluten Geschwindigkeiten bestimmt wird, benötigen wir noch einen Sensor, der die Eigengeschwindigkeit des Fahrzeugs liefert.

1.3.2 Prozessor

Der Radarsensor liefert stets die Entfernung des nächst gelegenen Objekts zum Sensor. Da aber nur Objekte auf der benachbarten Spur interessant sind, muss der Prozessor alle Entfernungsmessungen, die einen für jeden Sektor bestimmten Wert überschreiten, verwerfen.

Dadurch, dass der Radarsensor einen Bereich zum Beispiel schräg hinter dem Fahrzeug überwacht, stimmt die gemessene Entfernung nicht mit der wirklichen Entfernung überein. Der Prozessor muss diesen Fehler korrigieren.

Da sich der Sicherheitsabstand nach hinten nicht aus der eigenen Geschwindigkeit ergibt, muss die Geschwindigkeit hier berechnet werden. Dazu berechnet der Prozessor aus zwei Entfernungsmessungen die relative Geschwindigkeit zum eigenen Fahrzeug. Um die absolute Geschwindigkeit zu erhalten muss die eigene Geschwindigkeit addiert werden.

Nach vorn entfällt die Geschwindigkeitsberechnung, da sich nach vorn der Sicherheitsabstand aus der eigenen Geschwindigkeit berechnet.

Anhand der ermittelten Geschwindigkeiten und Sicherheitsabstände kann der Prozessor nun berechnen, ob ein sicherer Spurwechsel möglich ist.

1.3.3 Benutzerschnittstelle

Die Benutzerschnittstelle fällt sehr einfach aus, da keine Eingaben vom Benutzer nötig sind. Dem Benutzer wird lediglich angezeigt, ob er die Spur sicher wechseln kann.

Um den Fahrer nicht abzulenken, soll eine Warnung angezeigt werden, falls kein Spurwechsel möglich ist. Die Anzeige besteht aus einem Dreieck im Außenspiegel. Das Dreieck leuchtet durch die Spiegelfläche je nach Warnstufe in verschiedenen Farben.

2 Architekturf Entwurf und Mikroprozessor-Umgebung

2.1 logische Ebene

Als Ziel haben wir uns gesetzt einen möglichst effizienten Prozessor zu entwerfen. Durch eine geringe Taktrate wird Strom gespart. Dies ist nur möglich, wenn die Takte optimal genutzt werden. Deshalb haben wir zum Beispiel auch auf einen noop-Befehl verzichtet. Wenn der Prozessor auf Daten warten muss, wird der Takt angehalten. Unser System ist stark vom Sensor abhängig. Da sich der Radarsensor permanent dreht, aber nur bei einer halben Drehung Messdaten anfallen, hat unser System zwei Phasen. In der ersten Phase findet die Datengewinnung statt und in der zweiten Phase die Verarbeitung mit anschließender Ausgabe. Eine Umdrehung des Sensors dauert 1/10 Sekunde. Das nur in Phase zwei eine Warnung ausgegeben wird, stellt also kein Problem dar.

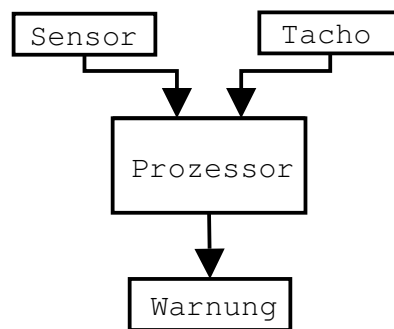


Abbildung 4: Architekturschema

2.1.1 Datengewinnung

Eine Umdrehung des Radarsensors ist in fünf Bereiche unterteilt:

Der erste geht von 0° bis 30° , wobei 0° der eigenen Fahrtrichtung entsprechen. Da der Sensor nur 2° pro Messung erfasst, ist der erste Bereich in 15 Sektoren unterteilt. Für jeden dieser Sektoren wird überprüft, ob die gemessene Entfernung nicht die maximal relevante Entfernung überschreitet und falls nein gespeichert.

Der zweite Bereich geht von 30° bis 90° . Innerhalb dieser 30 Sektoren wird nur

geprüft, ob die gemessene Entfernung nicht die maximal relevante Entfernung überschreitet. Wenn das nicht der Fall ist, dann befindet sich ein Fahrzeug unmittelbar neben uns.

Zur Initialisierung ist es noch nötig, dass der Prozessor weiß, wann der Radarsensor im ersten Sektor ist.

Für den nach hinten benötigten Sicherheitsabstand muss die Geschwindigkeit des nach dem Spurwechsel hinter dem eigenen Fahrzeug Fahrenen bestimmt werden. Dazu wird aus zwei Entfernungsmessungen die relative Geschwindigkeit zum eigenen Fahrzeug errechnet und zur absoluten Geschwindigkeit des eigenen Fahrzeugs addiert.

2.1.2 Datentypen und -formate

Als Daten treten Entfernungen und Geschwindigkeiten, sowie Sektornummern auf. Da eine Arithmetisch-Logische-Einheit (ALU) Ganzzahlen schneller verarbeiten kann als Fließkommazahlen, haben wir uns für ein Ganzzahlformat entschieden. Um trotzdem nicht an Genauigkeit zu verlieren, verarbeiten und speichern wir die Entfernungen in Zentimeter, die Geschwindigkeiten in Zentimeter pro Sekunde. Da nur 90 Sektoren existieren, muss hier nicht vereinfacht werden.

Die größtmögliche Zahl ergibt sich aus der größtmöglichen Entfernung von 15000cm. Da bei der Berechnung auch negative Zahlen verarbeitet werden müssen, stellen wir alle Zahlen im 16Bit-Zweierkomplement dar.

Zur Entfernungskorrektur und zur Bestimmung der maximal relevanten Entfernung in einem Sektor werden allerdings Winkelfunktionen benötigt. Die Berechnung der maximal relevanten Entfernungen lässt sich umgehen, indem die Werte in einer Tabelle nachgeschlagen werden. Dies hat den zusätzlichen Vorteil, dass die Arithmetisch-Logische-Einheit einfacher ausfällt.

Da die Entfernungskorrektur von der gemessenen Entfernung abhängt, lassen sich die Werte nicht so einfach zwischenspeichern. Hier wenden wir einen mathematischen Kniff an: Der eigentliche Cosinuswert wird von 1 subtrahiert und davon wird das Reziproke gespeichert. Damit können wir Ganzzahlen speichern und trotzdem

einen relativ genauen cosinus bestimmen.

Zusammengefasst haben wir nur ein Datenformat für alle Datentypen: das 16Bit-Zweierkomplement.

2.1.3 Verarbeitung und Speicherung

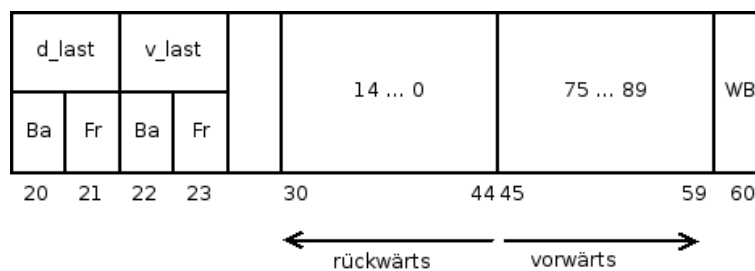


Abbildung 5: Der Datenspeicher

Zur Laufzeit sind im Speicher das Assembler-Programm und die zwei Lookup-Tabellen abgelegt. Diese müssen allerdings beim Systemstart erstmal aus einem langsam aber nicht-flüchtigen Speicher (zB EEPROM) in den schnellen RAM geladen werden. Die eine Tabelle repräsentiert Cosinuswerte und die andere enthält die maximal relevanten Sektorlängen. Die Organisation des Datenspeichers während der Programmausführung ist in Abbildung 5 dargestellt. Da unser Programm 201 Datenworte umfasst, die Lookup-Tabellen zusammen 60 und wir dann auch noch Datenspeicher benötigen, reichen $256 * 16$ Bit (Wortbreite) Speicher leider nicht aus. Deswegen benötigen wir 1kByte ($512 * 16$ Bit) CPU-nahen (weil schnellen) RAM.

2.1.4 Datenausgabe

Die Datenausgabe fällt bei unserem Spurwechsellassistent sehr einfach aus. Es wird eine 2Bit Steuerleitung nach außen geführt. Dadurch wird gesteuert, ob das Dreieck im Außenspiegel gelb, orange, rot oder gar nicht leuchtet.

2.2 technische Ebene

Die Hardware besteht aus dem Radarsensor mit Winkelmesser, dem Prozessor und einem Geschwindigkeitsmesser.

2.2.1 Schnittstellen

Für jede Hardware außerhalb des Prozessors muss eine Schnittstelle definiert werden.

Die wichtigste Schnittstelle ist die zwischen Sensor und Prozessor:

Der Sensor setzt das Signal 'Sensor Fetch Complete' (SFC), wenn neue Daten bereitstehen und der Prozessor den aktuellen Sektor und dessen Entfernungsmessung übernehmen kann. Da die Daten nicht gepuffert werden, muss der Prozessor die Daten sofort übernehmen. Die Datenleitungen für die Sektornummer und für die Entfernung ist 16Bit breit. Die Werte liegen im Zweierkomplement vor.

Desweiteren existiert noch eine Schnittstelle zwischen Prozessor und dem fahrzeug-internen Bus. Vom fahrzeug-internen Bus wird die eigene Geschwindigkeit des Fahrzeugs als 16Bit-Zweierkomplement abgefragt.

Zur Ausgabe einer Warnung existiert die Schnittstelle zwischen Prozessor und Leuchtdioden hinter dem Glas des Außenspiegels. Die Leuchtdioden, die als Dreieck leuchten, werden durch eine 2Bit Leitung angesteuert. Das Signal wird decodiert zu rot, orange, gelb oder nicht leuchten.

2.2.2 Sensorsteuerung

Der Sensor ist ein drehbarer Radarsensor, der Entfernungen misst. Im Sensor ist ein Winkelmesser integriert, der aus einem Rad mit zwei Lochreihen besteht. Die äußere Lochreihe hat alle 2° ein Loch. Die innere Lochreihe hat nur ein Loch, um den ersten Sektor zu markieren. Wenn die Lichtschranke auf der äußeren Lochreihe ein Loch erkennt, wird das Signal SFC generiert und der Sektorzähler um eins erhöht. Wenn die innere Lichtschranke das Loch erkennt, wird der Sektorzähler auf null gesetzt.

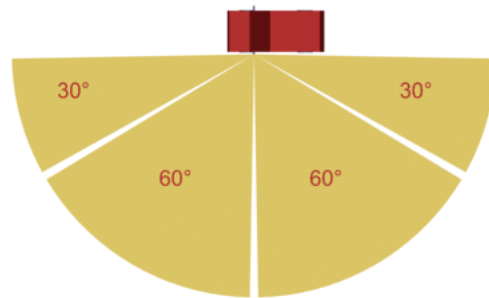


Abbildung 6: Sektoreinteilung

2.2.3 Speicherverwaltung

Das Assembler-Programm liegt im vordersten Teil des Speichers. Nach dem Programm kommen die beiden Tabellen. Auf diese kann durch die Flags MEMm und MEMc ohne größeren Aufwand von Seiten des Prozessors zugegriffen werden. Die Flags veranlassen das mit einer niedrigen Adresse im Befehlsaufruf, die Adressen hinter dem Programm angesprochen werden.

3 Mikroprozessor

3.1 Befehlsformat

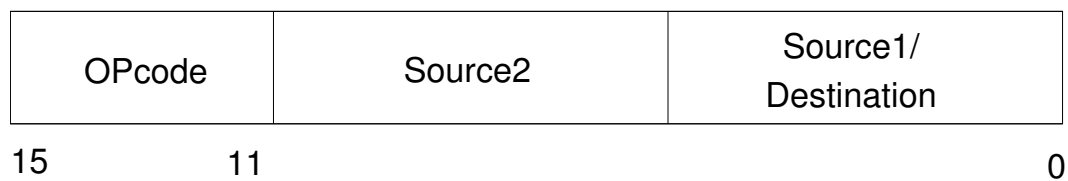


Abbildung 7: Das Befehlsformat

Ein Befehlswort ist 16Bit lang. Die vorderen 5Bit werden für den Opcode verwendet. Der Opcode bestimmt den Befehl und die Adressierungsart. Ein Aufspalten

von Opcode und Adressmodi ist nicht sinnvoll, da nur zu einem von 16 Befehlen alle Adressvarianten benötigt werden. Der Großteil der Befehle hat nur einen Adressmode.

Die Verbleibenden 11Bit enthalten mindestens ein Register (3Bit), da die grundlegende Architektur eine 1,5-Adressmaschine ist. Im Befehl kann noch ein weiteres Register (3Bit) enthalten sein. Die restlichen 4Bit bleiben leer. Falls im Befehl eine Konstante enthalten ist, muss ein zweites Datenwort geladen werden.

3.2 Befehlssatz

Für einige Befehle gibt es verschiedene Adressmodi, zum Beispiel für den MOV-Befehl. Das System ist eine 1,5-Adressmaschine, das heißt mindestens einer der Operanden ist ein Register. Es gibt die Möglichkeit der Register-direkten-Adressierung (R0-7), der Register-indirekten-Adressierung(@R0-7), und der absoluten Adressierung(@X), immediate-Adressierung(X) ist ebenfalls möglich. Bei der absoluten Adressierung und der Immediate-Adressierung wird der Operand im Datenwort nach dem entsprechenden Befehl gespeichert und muss nachgeliefert werden.

Der Befehl MOV kopiert Werte, entweder schreibt er angegebene Werte in ein Register oder kopiert den Inhalt von einer Speicherstelle bzw. Register in eine andere Speicherzelle, bzw ein anderes Register. Je nach Notwendigkeit von Speicherzugriffen, benötigt der MOV-Befehl unterschiedlich viel Zeit, abhängig von den verwendeten Adressierungsarten.

Es gibt mehrere Sprungbefehle. Zwei unbedingte Sprünge JMP und drei bedingte Sprünge JNZ, JGE und JL. JMP bekommt eine Register in dem die Sprungadresse angegeben ist oder JMP bekommt eine absolute Adresse. Die bedingten Sprünge benötigen ein Register und eine absolute Adresse. Der Inhalt des Registers wird mit Null verglichen, bei JNZ erfolgt der Sprung, wenn eine Gleichheit mit Null vorliegt. Wenn festgestellt wird das der Wert im Register größer oder gleich Null ist, wird bei JGE gesprungen und bei JL erfolgt der Sprung, wenn der Inhalt des Registers negativ ist.

Die Befehle INC R und DEC R inkrementieren bzw. dekrementieren den Inhalt

des Registers R. Die Arithmetisch-Logische-Einheit beherrscht außer den Vergleichen, der Inkrementierung und der Dekrementierung noch drei der Grundrechenarten, ADD, SUB und MUL. ADD kann den Inhalt von zwei Registern oder einen Immediate-Wert und den Inhalt eines Registers addieren. Beim Subtrahieren werden SUB entweder zwei Register oder ein Register und ein Immediate-Wert übergeben, wobei SUB A B bedeutet $B = B - A$. Bei MUL verhält es sich genauso nur das die Werte multipliziert werden.

Für die Berechnung der Warnstufe werden spezielle Befehle zur Verfügung gestellt. CORR wird benötigt um die reale Entfernung eines Objektes zu ermitteln und MAXL um festzustellen ob die maximal relevante Entfernung für den Sektor, der im angegebenen Register steht, zu ermitteln und in ein anderes Register zu schreiben. Da die eigene Geschwindigkeit benötigt wird, gibt es den Befehl GETV. GETV schreibt die aktuelle eigene Geschwindigkeit in das Register, das beim Befehlsaufruf mit angegeben werden muss. Um die Daten vom Sensor zu bekommen gibt es die Befehle GETS und GETD, wobei GETS den aktuellen Sektor in ein Register schreibt und GETD die Entfernung zum nächsten Objekt, im aktuellen Sektor, in ein Register schreibt.

Um das richtige Warnsignal zu setzen, brauchen wir den Befehl WARN, dieser schaltet je nach angegebenen Wert ein gelbes, oranges oder rotes Leuchtsignal.

3.3 Steuereinheit

Die Steuereinheit besteht aus dem Taktgeber (Takt), dem Taktunterbrecher, dem Adressgenerator und dem Mikroprogrammspeicher mit nachgeschaltetem Dekoder.

3.3.1 Steuereinheit

Der OPcode aus dem IR gibt dem Adressgenerator die erste Adresse eines Befehls im Mikroprogrammspeicher. Vom Mikroprogrammspeicher geht der Befehl in den Dekoder, dort wird der Mikrobefehl dekodiert und die entsprechenden Signale gesetzt um die einzelnen Teile des Prozessors anzusteuern. Der Taktunterbrecher, unterbricht den Takt für die Zeitspanne, die der Speicher bzw. der Sensor brauchen um ihre Daten bereit zu stellen, bzw. eine Zugriffsoperation abzuschließen. Im Adressgenerator

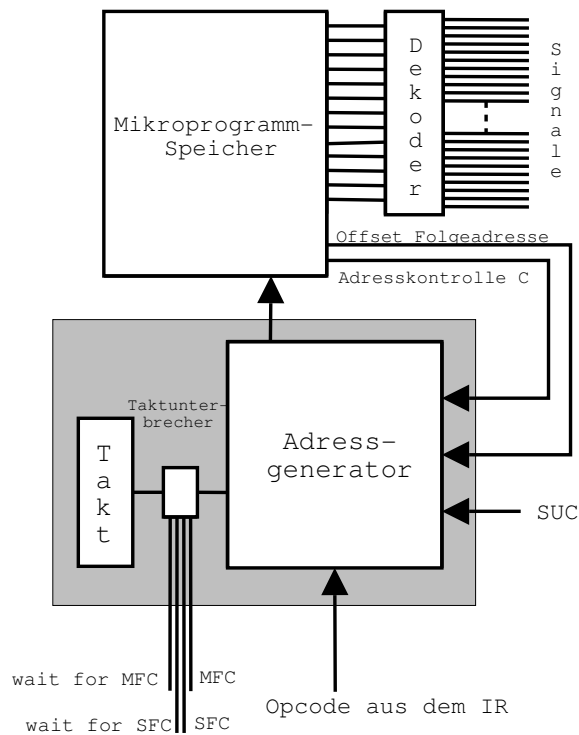


Abbildung 8: Die Steuereinheit

wird aus der aktuellen Adresse und einem Offset die Folgeadresse berechnet.

3.3.2 Mikroprogramm Speicher

Im Mikroprogramm Speicher sind die Mikrobefehle abgelegt. Ein Mikrobefehl ist 27Bit lang. Da unser Prozessor mit 51 verschiedenen Mikrobefehlen auskommt, benötigen wir weniger als 2kBit Mikroprogramm Speicher. Im Anhang ist der komplette Mikrocode mit Mikroprogramm Speicheradresse (Mpsa.), Opcode, Mnemonik des umgesetzten Assemblerbefehls, dem eigentlichen Mikrocode, der Folgeadresse (FA) im Mikroprogramm Speicher dem compute-Flag (cF) und der Beschreibung des Mikrocodes in Register-Transfer-Sprache (RTL).

3.3.3 Dekoder

Da die Mikrobefehle kodiert gespeichert werden, müssen sie vom Dekoder in die einzelnen Steuerleitungen dekodiert werden. Wir haben uns für eine Kodierung des Mikrobefehle entschieden, weil dadurch Mikroprogrammspeicher gespart werden kann. Allerdings haben wir uns auch gegen eine vertikale Kodierung entschieden, da dann die Dekodierung nicht parallel ablaufen kann.

Die Mikrobefehle sind in Gruppen zusammengefasst, in denen nur ein Steuersignal pro Mikrobefehl auftauchen kann. Daraus ergibt sich folgende Kodierung:

Breite	Steuersignal	Beschreibung
4Bit	8x Rn_{out} MDR_{out} PC_{out} ALU_{out} D_{out} S_{out} V_{out}	Für jeden Allzweckregister eine Leitung.
1Bit	$read$	
1Bit	$write$	
2Bit	W	Die Warnsignalleitung wird erst in der Anzeige dekodiert.
1Bit	$MEMm$	
1Bit	$MEMc$	
2Bit	$wait\ for\ MFC$ $wait\ for\ SFC$ C_{in}	
1Bit	ALU_{in}	
1Bit	T_{in}	
1Bit	CLR_T	
4Bit	8x Rn_{in} IR_{in} PC_{in} MAR_{in} MDR_{in}	Für jeden Allzweckregister eine Leitung.

3.3.4 Adressgenerator

Im Adressgenerator wird die Adresse des nächsten Mikroprogrammschritts bestimmt. Die Folgeadresse und das compute-Flag stehen im aktuellen Mikroprogrammbefehl. Wenn das compute-Flag gesetzt ist, dann wird zur Folgeadresse der opcode addiert. Ansonsten wird die Folgeadresse als nächste Adresse verwendet.

Das Steuersignal SUC wird durch die Arithmetisch-Logische-Einheit generiert. Wenn SUC gesetzt, ist ein Vergleich in der ALU wahr und es wird zum nächsten Assemb-

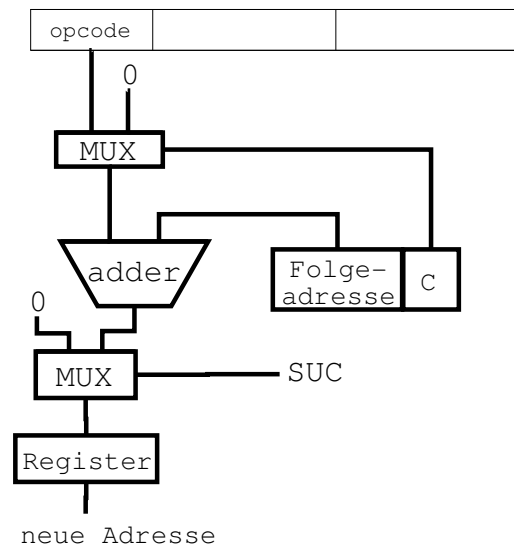


Abbildung 9: Der Adressgenerator

lerbefehl gesprungen. Die nächste Adresse im Mikroprogrammsspeicher ist dann 0, da ein neuer Assemblerbefehl geholt wird.

3.3.5 Taktunterbrecher

Der Taktunterbrecher ist nötig, da kein neuer Mikrobefehl ausgeführt werden soll, solange der Prozessor auf Daten aus dem Speicher oder vom Sensor wartet. Durch das unterbrechen des Taktes und den Verzicht auf einen noop-Befehl wird Strom und Speicherplatz im Mikrocode gespart.

MFC ist Eins, wenn der Speicher mit seiner Zugriffsoperation fertig ist. Das Flag wait for MFC ist gesetzt, wenn ein Speicherzugriff läuft und der Prozessor auf dessen Beendigung warten muss. Wenn beide Flags gleichzeitig gesetzt sind, geht der Takt weiter, ist nur wait for MFC gesetzt, hält der Takt und ist nur MFC gesetzt oder keins von beiden, wirkt die Negation von wait for MFC und der Takt geht normal weiter. Sollte von Seiten des Speichers keine Taktunterbrechung vorliegen, könnte das Warten auf den Sensor unseren Takt anhalten. Wenn wait for SFC gesetzt ist, wird

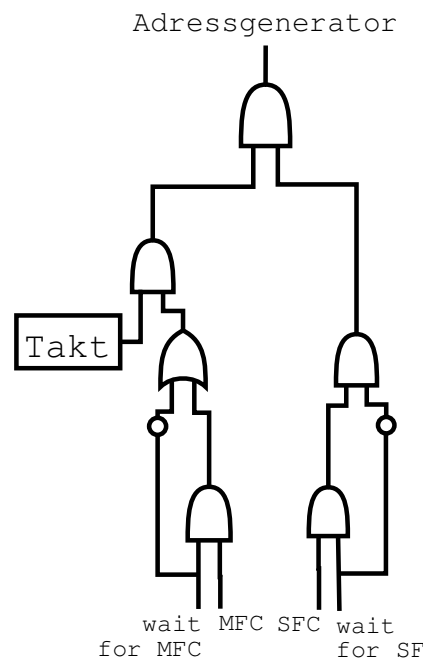


Abbildung 10: Der Taktunterbrecher

der Takt solange angehalten bis SFC auf Eins gesetzt wird. Sind beide Sensorenflags gesetzt, bzw. wait for SFC nicht, geht der Takt normal weiter.

3.4 Software

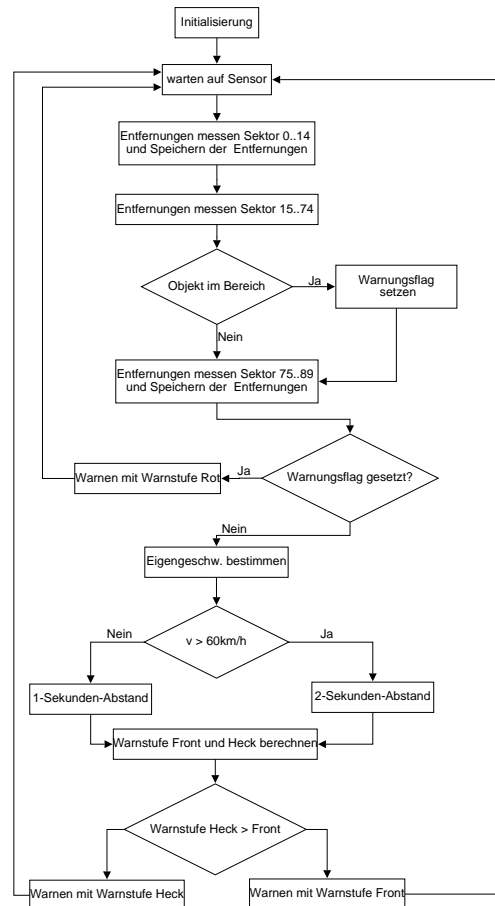


Abbildung 11: Programmablaufplan des Programms

Die Funktionsweise unseres Programms ist in Abbildung 11 zu sehen. Beim Start des Systems wird erstmal initialisiert. Gleich darauf treten wir schon in den Main-Loop des Programms ein. Es werden die Abstände zu anderen Fahrzeugen und Hindernissen in den einzelnen Sektoren bestimmt. Dabei wird zwischen *far* und *near* un-

terschieden. Im Nahbereich wird sofort mit der höchsten Warnstufe gewarnt, wenn ein Objekt im überwachten Bereich neben dem Fahrzeug ist. Eine Speicherung der Entfernung des entdeckten Objekts ist hierbei nicht nötig. Im Fernbereich werden die gemessenen Distanzen zwischengespeichert um aus jeweils 2 Distanzwerten eine Relativgeschwindigkeit des entdeckten Objekts zum eigenen Fahrzeug ermitteln zu können. Diese wird dann dazu verwendet zu berechnen, ob das entdeckte Objekt eine Gefahr für einen Spurwechsel darstellt oder nicht. Dazu wird der Sicherheitsabstand berechnet nach 1- oder 2-Sekundenregel berechnet. Je nachdem, ob wir uns innerorts oder außerorts befinden. Für Front- und Heckbereich werden unterschiedliche Sicherheitsabstände berechnet. Im Frontbereich wird der Sicherheitsabstand unseres Fahrzeugs abhängig von unserer Eigengeschwindigkeit bestimmt und im Heckbereich wird der Sicherheitsabstand des folgenden Fahrzeugs berechnet. Wenn der Sicherheitsabstand unterschritten wird, ist die Warnstufe Orange. Wird der Abstand in Kürze unterschritten (1 Sekunde innerorts und 2 Sekunden außerorts) ist die Warnstufe Gelb. Nachdem die Warnstufen für Front und Heck unabhängig bestimmt wurden, wird die größere Warnstufe ausgegeben. Das komplette Programm liegt in kommentierter Form im Punkt 5.1 vor.

3.5 Schaltungsentwurf

Es liegt ein 16 Bit Prozessor vor, alles in der CPU ist auf den Transport, bzw. die Verarbeitung von 16Bit Datenwörtern ausgelegt. Die Taktrate des Prozessors muss nur mindestens 1.8 KHz betragen und kann somit recht niedrig ausfallen. Es gibt acht Allzweck-Register (R0 bis R7), in denen berechnete und gemessene Werte bzw. Sprungadressen zwischengespeichert werden können. Desweiteren sind ein Programmzähler (PC) und ein Instruktions-Register (IR) vorhanden. Im PC befindet sich die Adresse zum nächsten Befehl im Speicher. Das IR hat eine Datenleitung zur Steuereinheit (CU) um den aktuellen Befehl an die CU zu geben, welche den Befehl auswertet und die entsprechenden Signale setzt.

Die Arithmetisch-Logische-Einheit (ALU) bekommt einen ihrer Werte aus dem temporären Register (T) und der andere ist in ihrem internen Register abgelegt. In T kann vom Bus eingelesen werden und nur von der ALU ausgelesen werden, da einige

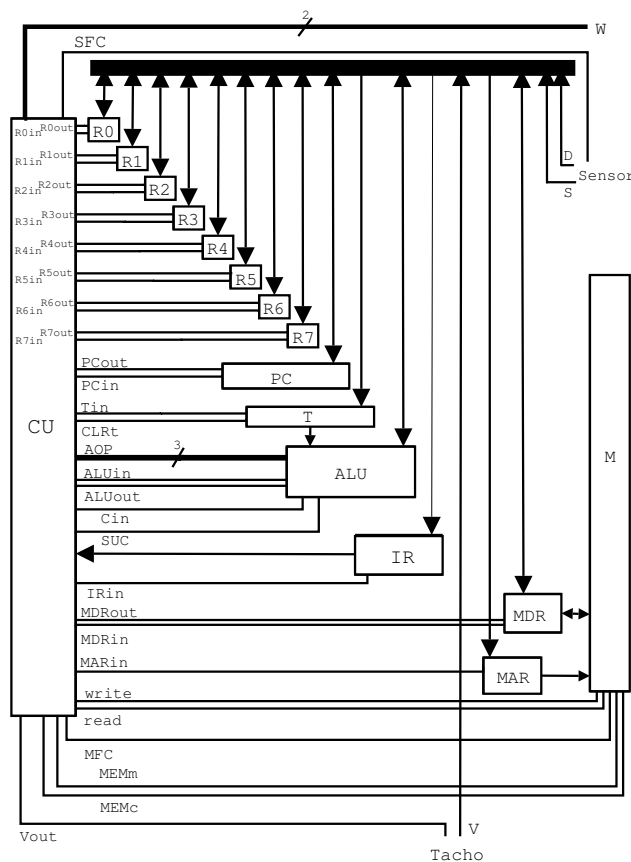


Abbildung 12: Die CPU

Befehle wie `inc`, nur einen Wert verwenden, kann mit Lösche T (`CLRt`) der gesamte Inhalt von T mit Nullen überschrieben werden. Die ALU bekommt ihr Instruktionen über eine 3Bit Leitung (`AOP`) von der CU, ausserdem kann `Cin` gesetzt werden, damit wird das C-Flag auf Eins gesetzt, sonst ist es immer Null. Die ALU hat ein Flag das der CU anzeigt ob ein Vergleich erfolgreich war (`SUC`), es ist Eins bei Erfolg und Null bei Misserfolg.

Für den Speicherzugriff stehen das Datenregister (`MDR`) und das Adressregister (`MAR`) zur Verfügung, in `MAR` befindet sich die Speicheradresse auf die zugegriffen wird und in `MDR` liegen die Daten die aus dieser Speicherstelle geladen wurden,

bzw. die dort reingeschrieben werden sollen. Der Speicher (M) wird mit den Signalen write und read angesprochen, write schreibt den Inhalt vom MDR an die Stelle die im MAR adressiert ist und read liest von der entsprechenden Adresse und legt den Inhalt im MDR ab.

Es gibt Schnittstellen zum Sensor, zum Tacho und zu den Warnleuchten. Zum Sensor gibt es drei Verbindungen, ein Flag das zeigt das der Sensor neue Daten hat (SFC), die Daten werden über S (Sektor) und D (Entfernung) auf den Bus gelegt. Das Warnsignal wird über eine 2Bit-Leitung angesprochen, wobei 01B gelbe Warnleuchte, 10B orangene und 11B rote Warnung bedeutet. Mit Vout wird die aktuelle Eigengeschwindigkeit auf den Bus geschrieben.

Von der CU gibt es für alle CPU-Register ein in-Signal (REGISTERNAMEin), dieses liest den Inhalt vom Bus in das entsprechende Register, für einige Register gibt es ein out-Signal (REGISTERNAMEout), dieses legt den Inhalt des Registers auf den Bus.

4 Schlussbetrachtungen

4.1 Einsatzmöglichkeiten

Der Spurwechsellassistent ist universell ausgelegt, sodass er in jedem Fahrzeugtyp egal ob PKW, Sattelschlepper oder Bus Anwendung finden kann. Der Mikrocode muss dann auf den Fahrzeugtyp angepasst werden. In dieser Arbeit wurde das System auf einen Durchschnitts-PKW angepasst.

Damit entsteht ein weltweites Anwendungsgebiet in der Automobilbranche.

Es ist auch denkbar den Spurwechsellassistenten in die Steuerung eines selbstfahrenden Fahrzeugs einzubauen.

4.2 Einschränkungen

Das größte Problem des Spurwechsellassistenten, so wie er hier vorgestellt wurde, ist, dass das System den Fahrbahnverlauf nicht erkennen kann. Als besonders schwer stellt sich das im Vorrauserkennen einer Kurve da. Damit funktioniert das System nur bedingt.

Für dieses Problem, dass auch bei Abstandshaltesystemen (adaptive cruise control ACC) auftaucht, sind bereits Lösungen in der entwicklung.

4.3 Sicherheitsbetrachtungen

Bevor der Spurwechselassistent großflächig eingesetzt werden kann, muss seine korrekte Funktionweise auch in Versuchen unter Beweis gestellt werden. Insbesondere die festgelegten Zeiten für die Dauer eines Spurwechsels müssen verifiziert werden. Obwohl Fehlwarnungen ungefährlich sind, sind auch sie unerwünscht, da sonst das System vom Käufer als nicht zuverlässig eingestuft wird. Ferner soll das System den Fahrzeugführer ja nicht stören und somit ablenken.

4.4 mögliche Erweiterungen

In Verbindung mit einem Spurhaltesystem, kann ein zusätzliches Warnsignal gegeben werden, wenn der Fahrer trotz Warnung durch den Spurwechselassistenten die Spur verlässt.

Zwingend nötig sind Sensoren zur Erkennung des Fahrbahnverlaufs. Wenn das Fahrzeug durch eine Kurve fährt, liefert der Radarsensor nicht mehr ausschließlich Informationen über die benachbarte Spur. Hinzu kommt, dass die benachbarte Spur vielleicht nicht vollständig einsehbar ist.

5 Anhang

5.1 Assemblercode

```
; .name MegaTama4 Software
;.author MegaTama4 Group
;.version 0.25
```

```
5 ;#####
; Typical Register Usage:
; R0: AccuMULator for Calculation / Address
```

```

; R1: Base Address (where is my data?)
; R2: Counter
10 ; R3: Data (Sector Distance)
; R4: Auxiliary Registers / Data2 (Sector Number: 0..89)
; R5: Auxiliary Registers
; R6: Garbage (meaning: I don't need this)
; R7: Jump Address (for return)
15 ;
; Operations:
; Format: op source2 source1/dest (SUB A B bedeutet B = B - A)
; Adressierungsarten:
; - Immediate: 42
20 ; - Absolute: @42
; - Register Direct: R3
; - Register Indirect: @R3
;
;         Datenrepraesentation:
25 ;         Entfernungen sind gespeichert in cm, Geschwindigkeiten in cm/s.
;
; Konstanten
;
; Die hoechstmoegliche Distanz, die gemessen werden kann:
30 ; QUASILINEAR = (270 / sin°(1)) * cos°(1), wobei 270 die Spurbreite
; in cm ist
; infinity = 15468
; sample_rate = 10
; insane_velocity = 10000
35 ; seconds_lane_change = 1 oder 2 (je nachdem ob innerorts oder
; ausserorts)
; 60 km/h = 16.67 m/s = 1667 cm/s
; city_velocity = 1667

```

```
40 ; Adressen
    ; front_data = 30
    ; back_data = 45
    ; WARN_buffer = 60
    ; last_distance = 20
45 ; last_velocity = 22

    ; Warning levels
    ; red = 3
    ; orange = 2
50 ; yellow = 1
    ; green = 0

    ;#####

    ; -- Main --
55 ; Init
    MOV 0 @20
    MOV 0 @21
    MOV 0 @22
    MOV 0 @23
60

    ; Main Loop
    main_loop:

    ;#####

65 ; -- Data Recording --
    ; Es gibt Sektoren im Nah- (near) und Fernbereich (far)
    ; far: 0..14, 75..89 (2 * 15 Sektoren vorne und hinten)
    ; near: 15..74 (60 Sektoren an der Seite)
    ; insgesamt: 90 Sektoren
70 ;
    ; Der Sensor arbeitet mit einer Frequenz von 10Hz, und jeder Sektor
```

```

; hat einen Bereich von °2.
; Dadurch ergibt sich eine Datenrate von 180 * 10 Hz = 1.8 KHz.
; Der Prozessor darf nicht langsamer sein als diese 1.8 KHz.
75
; auf Daten warten
    wait_for_sensor:
        GETS R4
        SUB 179 R4
80    JNZ R4 wait_for_sensor:

; Scanne die Sektoren 0..14 und 75..89 und schreibe ihre Werte in den
; Speicher.
; Scanne auch die Sektoren 15..74 und setze das Side Flag (SF),
85 ; wenn mindestens einer warnt.
; Die Worte 30..59 sind fuer die Distanzwerte der far-Sektoren
; reserviert:
; 30          45          6061 63
; +-----+-----+
90 ; |<_FRONT_FAR_><_BACK_FAR-->|  |
; |14    ...    075    ...    89|WB |
; +-----+-----+
;
; The Base Address is stored in R1.
95
; Sektoren 0..14 speichern
; R2 = 0..14, R1 laeuft rueckwaerts
    MOV 45 R1
    MOV 0 R2
100    front:
; hole Sensordaten
        GETD R3
; Teste, ob R3 im Bereich liegt

```

```

                                MAXL R2 R4
105                                SUB R3 R4
                                JGE R4 in_range_front:
; ausserhalb des Bereichs: auf infinity setzen
                                MOV 15468 R3
                                in_range_front:
110 ; schreibe Daten an vorherige Position
                                DEC R1
                                MOV R3 @R1
; bis R2 == 15
                                INC R2
115                                MOV R2 R0
                                SUB 15 R0
                                JNZ R0 front:

; Sektoren 15..44
120 ; R2 = 15..44
; R5 wird auf 1 gesetzt, wenn etwas im Bereich ist.
                                MOV 0 R5
                                side_first_half:
; hole Sensordaten
125                                GETD R3
; Teste, ob R3 im Bereich liegt
                                MAXL R2 R4
                                SUB R4 R3
                                JGE R3 out_of_range_fh:
130 ; im Bereich: Flag setzen
                                MOV 1 R5
                                out_of_range_fh:
; bis R2 == 44
                                INC R2
135                                MOV R2 R0
```

```

        SUB 44 R0
        JNZ R0 side_first_half:

; Sektoren 45..74
140 ; R2 = 44..15
        side_second_half:
; hole Sensordaten
        GETD R3
; Teste, ob R3 im Bereich liegt
145        MAXL R2 R4
        SUB R4 R3
        JGE R3 out_of_range_sh:
; im Bereich: Flag setzen
        MOV 1 R5
150        out_of_range_sh:
; bis R2 == 15
        DEC R2
        MOV R2 R0
        SUB 15 R0
155        JNZ R0 side_second_half:

; Sektoren 75..89
; R2 = 14..0, R1 laeuft vorwaerts
        MOV 45 R1
160        back:
; hole Sensordaten
        GETD R3
; Teste, ob R3 im Bereich liegt
        MAXL R2 R4
165        SUB R3 R4
        JGE R4 in_range_back:
; ausserhalb des Bereichs: auf infinity setzen
```



```
200      MOV 30 R1
        MOV 15 R2
        MOV 1 R5
        MOV got_front: R7
        JMP calc_WARN:
205 got_front:
        MOV R1 @60

        ; Berechne Warnung des Hecks
        MOV 45 R1
210      MOV 15 R2
        MOV 0 R5
        MOV got_back: R7
        JMP calc_WARN:
        got_back:
215      MOV @60 R0

        ; R0 = Warnung hinten , R1 = Warnung vorne
        ; R1 = max(R0, R1)
        ; R0 = R0 - R1
220      SUB R1 R0
        ; wenn R0 < R1: nichts tun
        JL R0 WARN_B:
        ; sonst: R1 = R1 + R0 (R1 = R0 auf Umweg)
        ADD R0 R1
225 WARN_B:
        WARN R1

        JMP main_loop:

230 ;#####
        ; - Warnung -
```

```
; Berechne Minimum der (korrigierten) Werte [R1] .. [R1+R2-1],
; daraus Geschwindigkeit, Sicherheitsabstand und Warnstufe
; R1 = Datenoffset, R2 = Datenlaenge, R4 = v_self,
235 ; R6 = Sicherheits-sekunden
; Resultat: Warnstufe in R1
calc_WARN:
; minimalen Wert (d) berechnen und in R0 speichern
    MOV 15468 R0
240 ; durchlaufe Daten
    data_loop:
        DEC R2
; Daten holen, R1 erhoehen
        MOV @R1 R3
245        INC R1
; Wichtig: R3 korrigieren
        CORR R2 R3
; falls R3 < R0: R0 = R3
        SUB R0 R3
250        JGE R3 greater:
        ADD R3 R0
    greater:
        JNZ R2 data_loop:

255 ; letzte Distanz laden
        MOV 20 R1
        ADD R5 R1
; Distanz speichern fuer Vergleich
        MOV @R1 R3
260        MOV R0 @R1

; v = (d_last - d) * f
; R3 = (R3 - R0) * sample_rate
```

```
                SUB R0 R3
265             MUL 10 R3
                ; R3 ist jetzt v in cm/s
                ; R3 positiv: Auto naehert sich
                ; R3 negativ: Auto entfernt sich

270             ; Ignorieren, falls |v| > insane_velocity
                ; R2 = R3.abs
                MOV R3 R2
                JGE R2 v_positive:
                MUL -1 R2
275 v_positive:
                ; letzte Geschwindigkeit laden
                MOV 22 R1
                ADD R5 R1
                SUB 10000 R2
280             JL R2 valid:
                ; zuruecksetzen, falls ungueltig
                MOV @R1 R3
                valid:
                ; Geschwindigkeit speichern
285             MOV R3 @R1

                ; Front: R4 = d_secure = v_self * R6
                ; Back: R4 = d_secure = (v_self + v) * R6
                JNZ R5 secure_self:
290             ADD R3 R4
                secure_self:
                MUL R6 R4

                ; R3 = R3 * seconds_lane_change
295             ; R3 = Distanzaenderung waehrend eines SpW
```

```
; R5 = d_soon = R0 - R3 = d - v * seconds_lane_change
; seconds_lane_change abhaengig davon,
; ob innerorts oder ausserorts 1 oder 2 Sekunden
; Entscheidung anhand von R6 (Sicherheitsekunden ; innerorts 1,
300 ; ausserorts 2)
    DEC R6
    JNZ R6 city:
    MUL 2 R3                ;seconds_lane_change = 2
city:
305    MOV R0 R5
    SUB R3 R5

; R5 = d_soon , R4 = d_secure
; Entscheide ueber Warnung
310
; GRUEN wenn keine andere Warnung
    MOV 0 R1

; ORANGE wenn d <= d_secure
315    SUB R4 R0
    JGE R0 no_WARN_orange:
    MOV 2 R1
no_WARN_orange:

320 ; GELB wenn d_soon <= d_secure
    SUB R4 R5
    JGE R5 no_WARN_yellow:
    MOV 1 R1
no_WARN_yellow:
325
JMP R7
```

5.2 Mikrocode

Mpsa	opcode	Befehl	Mikrocode	FA	cF	RTL
0		fetch	PCout, MARin, read, ALUin, CLRt, Cin, AOP=100	1		MAR←PC, ALU←PC
1			ALUout, PCin, wait for MFC	2		PC←ALU
2			MDRout, IRin	3	1	IR←MDR
3	00000	movRR	Rn1out, Rn2in	0		Rn2←Rn1
4	00001	movR@R	Rn1out, MDRin	28		MDR←Rn1
28			Rn2out, MARin, write, wait for MFC	0		Rn2←MAR
5	00010	movR@X	PCout, MARin, read, wait for MFC	29		MAR←PC
29			MDRout, MARin	30		MAR←MDR
30			Rn1out, MDRin, write, wait for MFC	0		MDR←Rn1
6	00011	mov@RR	Rn1out, MARin, read, wait for MFC	31		MAR←Rn1
31			MDRout, Rn2in	0		Rn2←MDR
7	00100	mov@XR	PCout, MARin, read, ALUin, CLRt, Cin, AOP=100	32		MAR←PC, ALU←PC
32			ALUout, PCin, wait for MFC	33		PC←ALU
33			MDRout, MARin, read, wait for MFC	31		MAR←MDR
8	00101	movXR	MDRout, Rn2in			Rn2←MDR
			PCout, MARin, ALUin, CLRt, Cin, AOP=100, read	51		MAR←PC, ALU←PC
51			ALUout, PCin, wait for MFC	31		PC←ALU
			MDRout, Rn2in			MDR←Rn2

Mpsa	opcode	Befehl	Mikrocode	FA	cF	RTL
9	00110	jmpX	PCout, MARin, read, wait for MFC	34		MAR←PC
34			MDRout, PCin	0		PC←MDR
10	00111	jmpR	Rn1out, PCin	0		PC←Rn1
11	01000	jnzRX	Rn1out, ALUin, CLRt, AOP=001	9		ALU←Rn1
			PCout, MARin, read, wait for MFC			MAR←PC
			MDRout, PCin			PC←MDR
12	01001	jgeRX	Rn1out, ALUin, CLRt, AOP=010	9		ALU←Rn1
			PCout, MARin, read, wait for MFC			MAR←PC
			MDRout, PCin			PC←MDR
13	01010	jlRX	Rn1out, ALUin, CLRt, AOP=011	9		ALU←Rn1
			PCout, MARin, read, wait for MFC			MAR←PC
			MDRout, PCin			PC←MDR
14	01011	incR	Rn1out, ALUin, CLRt, Cin, AOP=100	35		ALU←Rn1
35			ALUout, Rn1in	0		Rn1←ALU
15	01100	decR	Rn1out, ALUin, CLRt, Cin, AOP=101	35		ALU←Rn1
			ALUout, Rn1in			Rn1←ALU

Mpsa	opcode	Befehl	Mikrocode	FA	cF	RTL
16	01101	addRR	Rn1out, Tin, AOP=100	36		$T \leftarrow Rn1$
36			Rn2out, ALUin	37		$ALU \leftarrow Rn2$
37			ALUout, Rn2in	0		$Rn2 \leftarrow ALU$
17	01110	addXR	PCout, MARin, ALUin, CLRt, Cin, AOP=100, read	38		$MAR \leftarrow PC,$ $ALU \leftarrow PC$
38			ALUout, PCin, AOP=100	39		$PC \leftarrow ALU$
39			Rn2out, ALUin, wait for MFC	40		$ALU \leftarrow Rn2$
40			MDRout, Tin	41		$T \leftarrow MDR$
41			ALUout, Rn2in	0		$Rn2 \leftarrow ALU$
18	01111	subRR	Rn1out, Tin, AOP=101	36		$T \leftarrow Rn1$
			Rn2out, ALUin			$ALU \leftarrow Rn2$
			ALUout, Rn2in			$Rn2 \leftarrow ALU$
19	10000	subXR	PCout, MARin, ALUin, CLRt, Cin, AOP=100, read	42		$MAR \leftarrow PC,$ $ALU \leftarrow PC$
42			ALUout, PCin, AOP=101	43		$PC \leftarrow ALU$
43			Rn2out, ALUin, wait for MFC	40		$ALU \leftarrow Rn2$
			MDRout, Tin			$T \leftarrow MDR$
			ALUout, Rn2in			$Rn2 \leftarrow ALU$
20	10001	mulRR	Rn1out, Tin, AOP=110	36		$T \leftarrow Rn1$
			Rn2out, ALUin			$ALU \leftarrow Rn2$
			ALUout, Rn2in			$Rn2 \leftarrow ALU$
21	10010	mulXR	PCout, MARin, ALUin, CLRt, Cin, AOP=100, read	44		$MAR \leftarrow PC,$ $ALU \leftarrow PC$
44			ALUout, PCin, AOP=110	45		$PC \leftarrow ALU$
45			Rn2out, ALUin, wait for MFC	40		$ALU \leftarrow Rn2$
			MDRout, Tin			$T \leftarrow MDR$
			ALUout, Rn2in			$Rn2 \leftarrow ALU$

Mpsa	opcode	Befehl	Mikrocode	FA	cF	RTL
22	10011	maxlRR	Rn1out, MARin, MEMm, read, wait for MFC	9		MAR←Rn1
23	10100	corrRR	MDRout, Rn2in			Rn2←MDR
46			Rn1out, MARin, MEMc, read	46		MAR←Rn1
47			Rn2out, ALUin, AOP=111, wait for MFC	47		ALU←Rn2
48			MDRout, Tin	48		T←MDR
48			ALUout, Tin, AOP=101	36		T←ALU
			Rn2out, ALUin			ALU←Rn2
			ALUout, Rn2in			Rn2←ALU
24	10101	getsR	wait for SFC	49		
49			Sout, Rn1in	0		Rn1←S
25	10110	getdR	wait for SFC	50		
50			Dout, Rn1in	0		Rn1←D
26	10111	getvR	Vout, Rn2in	0		Rn2←V
27	11000	warnX	W=IR[0..1]	0		

6 Quellen

- Andrew S. Tanenbaum: „Structured Computer Organization“ Third Edition
- <http://www.informatik.uni-ulm.de/rs/projekte/core/prozessorarchitektur/script/K4.pdf>
- <http://www.informatik.uni-ulm.de/rs/projekte/core/prozessorarchitektur/script/K5.pdf>
- Skript zur Vorlesung Technische Informatik 2, Prof. Malek Sommersemester 2005