Prüfung
„Grundlagen der modernen Informationstechnik“
Sommersemester 2021

23.08.2021

durchgeführt via TUMexam

Musterlösung
- ohne Gewähr -

Aufgabe 1: Zahlensysteme

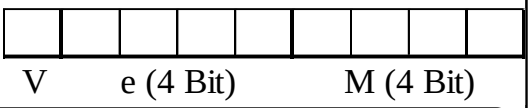
Überführen Sie die Zahl $(7243)_8$ des Oktalsystems in das Binärsystem.

$(7243)_8$ = (**111 010 100 011**)$_2$

Aufgabe 2: IEEE 754 Gleitkommazahlen

Rechnen Sie die Dezimalzahl $(-14{,}25)_{10}$ in eine Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) mit folgender Formatierung um:

Hinweis: Es gilt big endian.



Vorzeichenbit V = **1**

Mantisse M_2 (Binärzahl sowie normalisiert):

M_2 = **$(1110, 01)_2$** **=** **$(1{,}11001*2^3)_2$**

Exponent = **3**

Bias = **$2^{(x-1)}-1 = 2^{(4-1)}-1 = 7$**

Biased Exponent = **$B + E = (7+3)_{10} = (10)_{10} = (1010)_2$**

Vollständige Gleitkommazahl nach gegebener Formatierung:

1 1010 1100

Aufgabe 3: Hamming-Distanz

Bestimmen Sie die Hamming Distanz h für die zwei Codewörter 1001 und 1100.

2

Wie viele fehlerhafte Bits können mit einer Hamming-Distanz von h=2 gefunden und wie viele können korrigiert werden?

Anzahl Bits, die gefunden werden: **$e = h - 1 = 2 - 1 = 1$**

Anzahl Bits die korrigiert werden: **$f = (h - 1)/2 = 1/2 = 0$**

Aufgabe 4: Logische Schaltungen und Schaltbilder

a) Gegeben sei die Wahrheitstabelle (Tabelle 4.1). Erstellen Sie das zugehörige Schaltbild der **vollständigen** Disjunktiven Normalform (**DNF**). Statt ≥ können Sie auch >= schreiben.

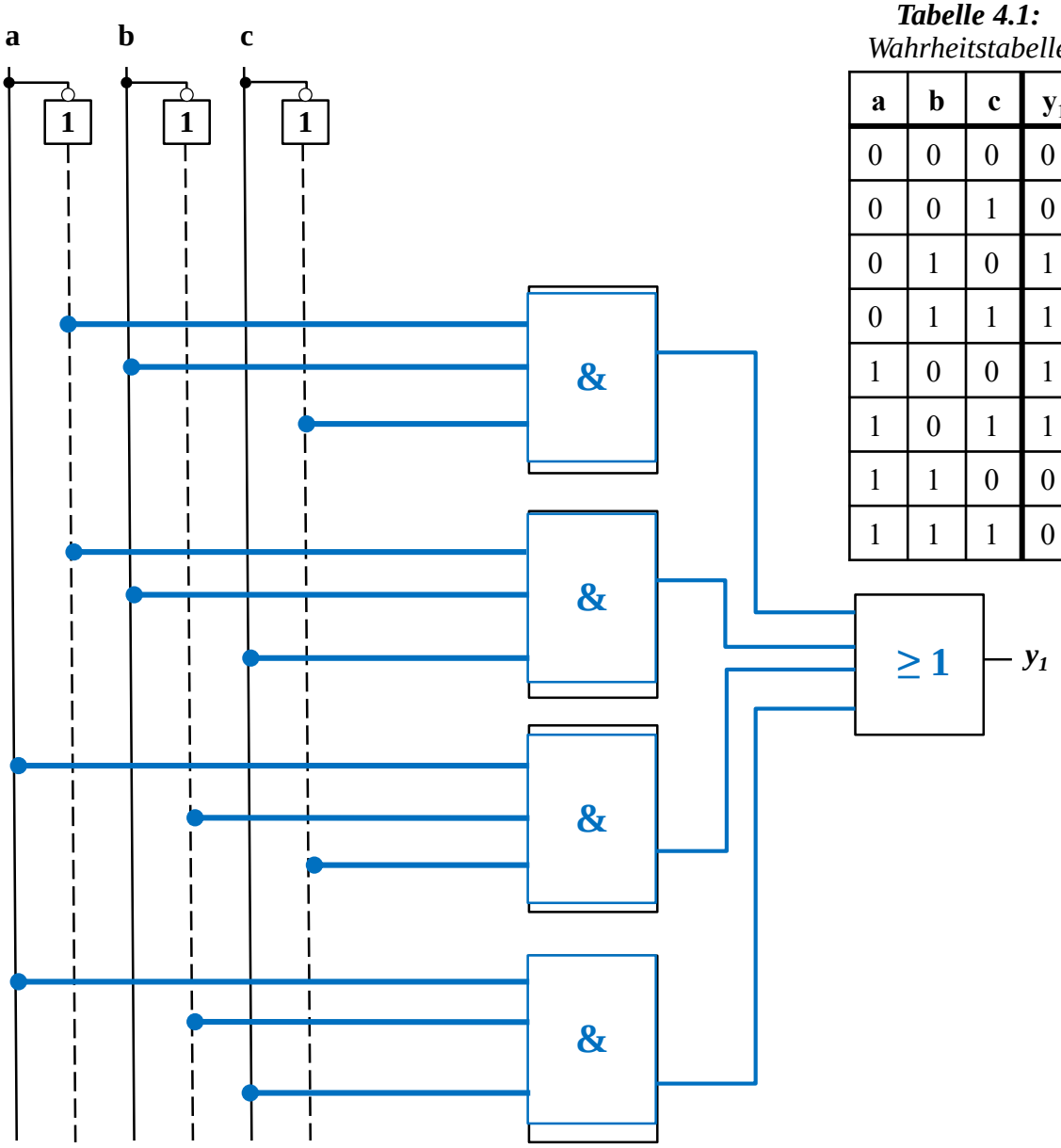


Tabelle 4.1:
Wahrheitstabelle

a	b	c	y_1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

b) Welche Schaltung ist in a) dargestellt? Bitte kreuzen Sie den richtigen Fachbegriff an.

( ) NAND-Gatter für 3 Eingänge

( ) Zweikanal-Demultiplexer mit Selektionseingang „c“ und Dateneingang y

( ) Zweikanal-Multiplexer mit Selektionseingang „b“

(X) XOR-Gatter für Eingänge „a“ und „b“

Aufgabe 5: Flip-Flops

Gegeben ist die folgende Master-Slave-Flip-Flop-Schaltung (Bild 5.1).

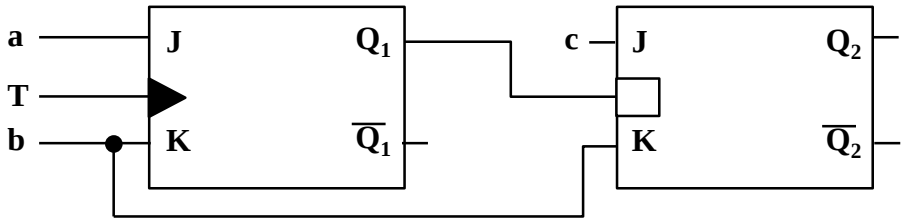
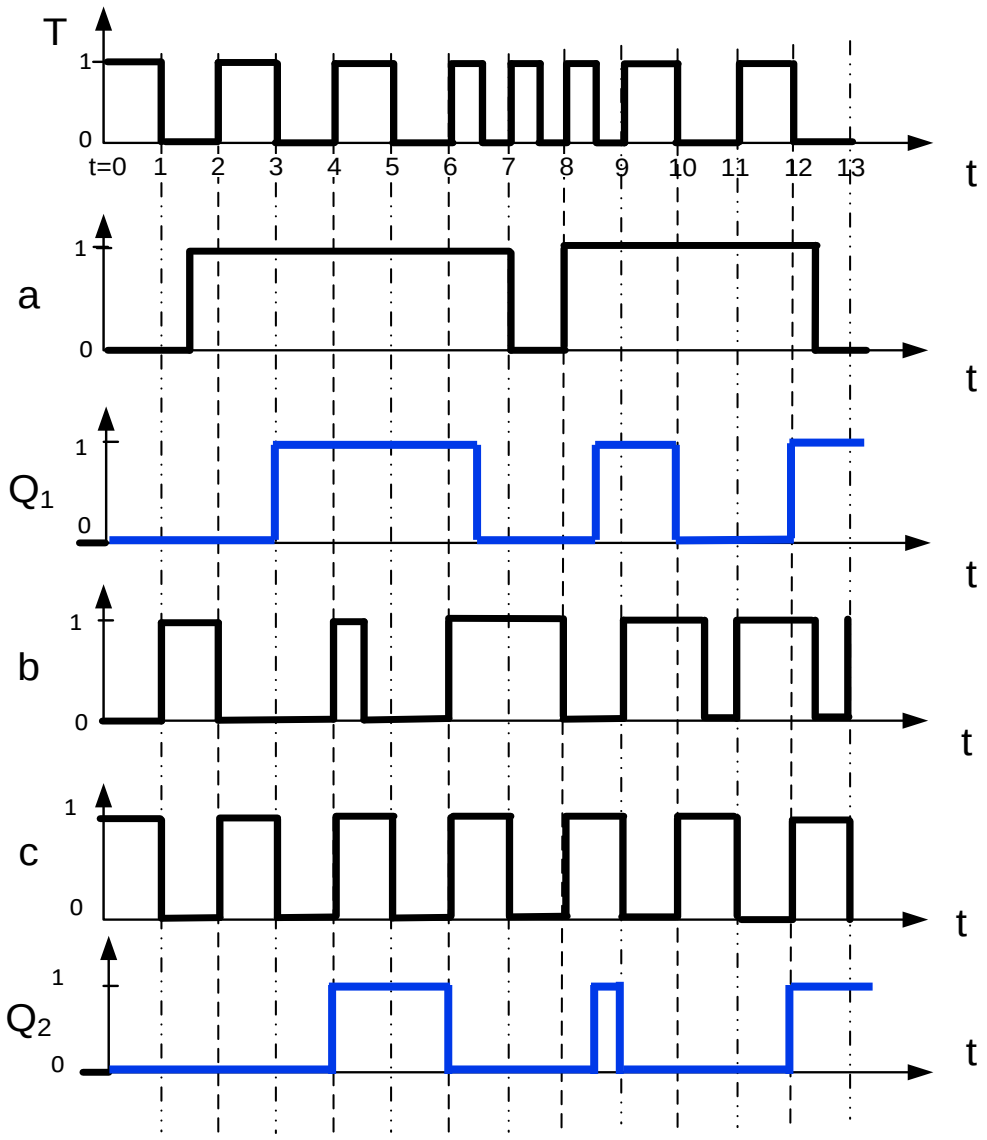


Bild 5.1: *MS-FF*

Zu Beginn (t = 0) sind die Flip-Flops in folgendem Zustand: $Q_1 = Q_2 = 0$.

Analysieren Sie die Schaltung für den Bereich t=[0;13], indem Sie für die Eingangssignale a, b, c und T die zeitlichen Verläufe für Q_1 und Q_2 in die vorgegebenen Koordinatensysteme eintragen.

Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden.



Aufgabe 6: MMIX Rechner

Gegeben sei der nachfolgende Algorithmus sowie ein Ausschnitt der MMIX-Code-Tabelle (Bild 6.1) und eines Registerspeichers (Bild 6.2):

	0x_0	0x_1	…	0x_4	0x_5	…
	0x_8	0x_9		0x_C	0x_D	
…	…	…	…	…	…	…
0x1_	FMUL	FCMPE	…	FDIV	FSQRT	…
	MUL	MUL I		DIV	DIV I	
0x2_	ADD	ADD I	…	SUB	SUB I	…
	2ADDU	2ADDU I		8ADDU	8ADDU I	
…	…	…	…	…	…	…
0x8_	LDB	LDB I	…	LDW	LDW I	…
	LDT	LDT I		LDO	LDO I	
0x9_	LDSF	LDSF I	…	CSWAP	CSWAP I	…
	LDVTS	LDVTS I		PREGO	PREGO I	
0xA_	STB	STB I	…	STW	STW I	…
	STT	STT I		STO	STO I	
…	…	…	…	…	…	…
0xE_	SETH	SETMH	…	INCH	INCMH	…
	ORH	ORMH		ANDNH	ANDNMH	

Bild 6.1: *MMIX-Code-Tabelle*

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x86	0x00 00 00 00 00 00 01 01	Nicht veränderbar
$0x87	0x00 00 00 00 00 00 00 06	Variable a
$0x88	0x00 00 00 00 00 00 00 01	Variable b
$0x89	0x00 00 00 00 00 00 00 01	Variable c
$0x8A	0x00 00 00 00 00 00 62 05	Zwischenergebnis
$0x8B	0x00 00 00 00 00 00 61 0B	Nicht veränderbar
…	…	

Bild 6.2: *Registerspeicher*

Im Registerspeicher eines MMIX-Rechners befinden sich zu Beginn die in Bild 6.2 gegebenen Werte. In der Spalte *Kommentar* wurde angegeben, welche Daten diese enthalten und wofür die einzelnen Zellen benutzt werden müssen.

Bearbeiten Sie die Aufgaben auf der Folgeseite.

Hinweis: Algorithmus für Aufgabenteil a):

Algorithmus (Dezimal): $\frac{15 * a + 257 - b}{b * 5}$

a) Führen Sie den rechts gegebenen Algorithmus aus. Übersetzen Sie diese Operationen in **Assembler-Code** mit insgesamt maximal 5 Anweisungen. Verwenden Sie dazu lediglich die in Bild 6.1 umrahmten Befehlsbereiche. Speichern Sie die Zwischenergebnisse nach jedem Befehl des Algorithmus nur in der Registerzelle mit dem Kommentar *Zwischenergebnis*.

1	MULI $0x8A $0x87 0x0F
2	ADD $0x8A $0x8A $0x86
3	SUB $0x8A $0x8A $0x88
4	DIV $0x8A $0x8A $0x88
5	DIVI $0x8A $0x8A 0x05

1	MULI $0x8A $0x87 0x0F
2	ADD $0x8A $0x8A $0x86
3	DIV $0x8A $0x8A $0x88
4	SUBI $0x8A $0x8A 0x01
5	DIVI $0x8A $0x8A 0x05

1	MULI $0x8A $0x87 0x0F
2	INCL $0x8A 0x01 0x01
3	SUB $0x8A $0x8A $0x88
4	DIV $0x8A $0x8A $0x88
5	DIVI $0x8A $0x8A 0x05

b) Angenommen Ihr Zwischenergebnis sei 0x0...00 12 34 56. Sie speichern es mit dem Befehl STW $0x8A $0x8B $0x86 in den Datenspeicher (Bild 6.3). Die Byte-Reihenfolge ist mit big endian festgelegt. Geben Sie für die vier Eingangsbusse am Datenspeicher an, welcher Wert dort jeweils anliegt. Tragen Sie die durch den Speicherbefehl geänderten Werte in den Datenspeicher ein.

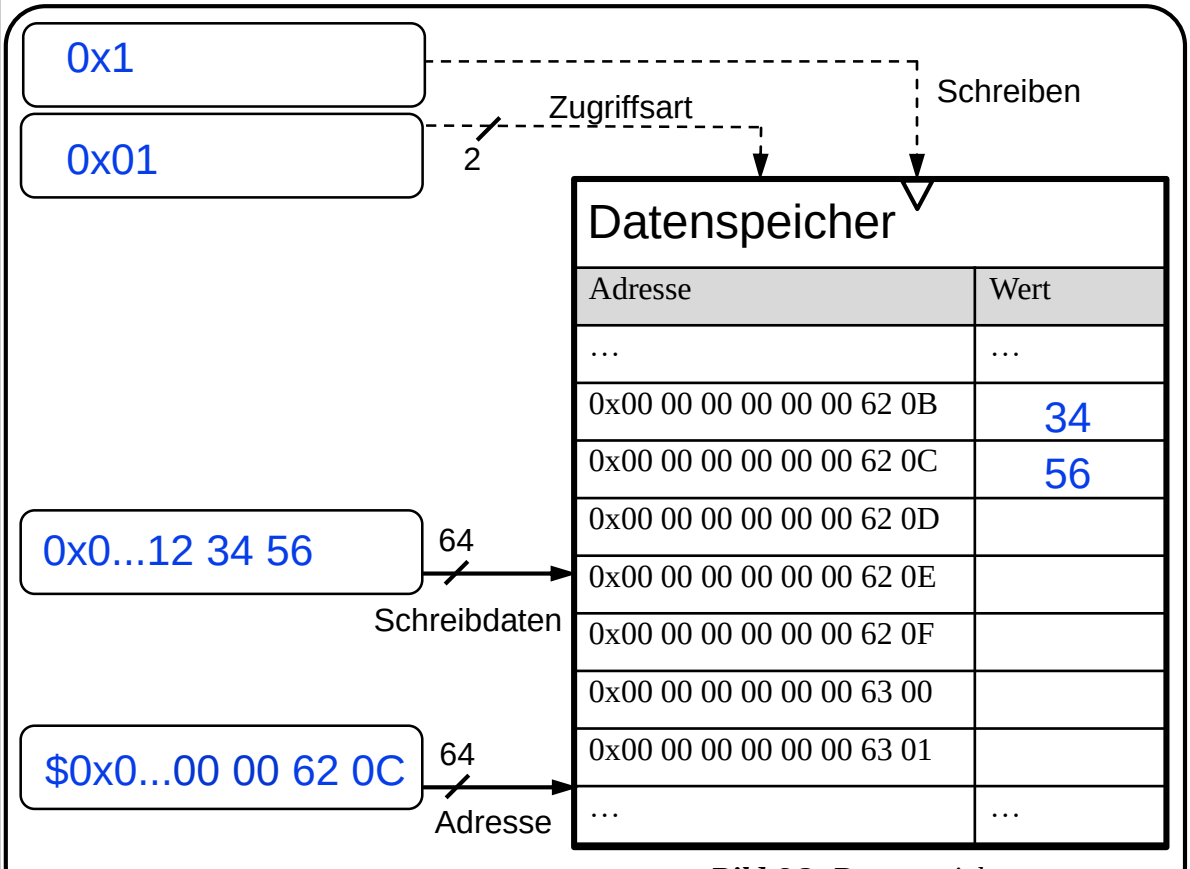


Bild 6.3: *Datenspeicher*

c) Geben Sie den Befehl M_4[$0x8A + 0xFF] = $0x88 in Assemblersprache an.

STTI $0x88 $0x8A 0xFF

Aufgabe 7: Scheduling

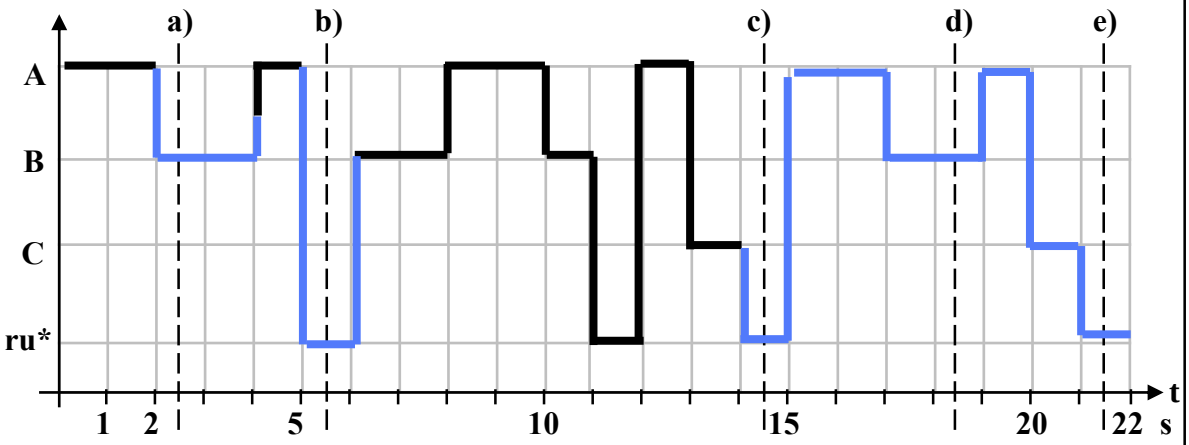
Auf einem Einprozessorsystem sollen die nachfolgend angegebenen Tasks einer Geschwindigkeitsüberwachung ablaufen.

Hinweis: Die Tabelle 7.1 enthält alle wichtigen Angaben zur Planung der Tasks A, B, C. Die Ausführungsdauer bestimmt die Dauer des Tasks. Die Tasks sollen ab dem Zeitpunkt Start eingeplant werden und die Wiederholrate gibt an, in welchem Abstand die Tasks relativ zum Start erneut aufgerufen werden.

Tabelle 7.1: *Übersicht über Tasks*

Bezeichnung	**Task**	Ausführungsdauer	Start	Wiederholrate
Datenlogging	A	3s	0s	alle 6s
Verkehrsanalyse	B	5s	1s	alle 14s
Blitzer	C	1s	13s und 20s	-

Gegeben ist das untenstehende Diagramm (Bild 7.1), das teilweise das präemptive Scheduling nach dem Schema Round-Robin für die Tasks A und B für den Zeitraum t = [0; 22] Sekunden zeigt. Ein Zeitschlitz hat eine Größe von zwei Sekunden. Task C ist ein Interrupt, der zweimal im Verlauf auftritt (vgl. „Start“ in Tabelle).



ru* = ruhend

Bild 7.1: *Taskverlauf nach präemptivem Scheduling*

Geben Sie an, welcher Task (A, B, C oder keiner d.h. „ruhend“) zu den folgenden Zeitpunkten aktiv ist.

a) **t = 2.5 :** **B**

b) **t = 5.5 :** **Ruhend**

c) **t = 14.5 :** **Ruhend**

d) **t = 18.5 :** **B**

e) **t = 21.5 :** **Ruhend**

Aufgabe 8: Semaphoren und Zeitparameter von Rechenprozessen

a) Gegeben seien die folgenden vier Tasks T1 bis T4 sowie die Semaphoren S1 bis S4 (Bild 8.1). Beantworten Sie die folgenden Fragen.

T1	T2	T3	T4
P(S1)	P(S2)	P(S2)	P(S4)
	P(S2)	P(S3)	P(S4)
…	...	…	...
V(S4)	V(S3)	V(S2)	V(S1)

Startwert:

S1	0
S2	3
S3	0
S4	0

Bild 8.1: *Semaphorenzuweisung und Startwerte*

Angenommen ein Task startet sofort, sobald die entsprechenden Semaphoren verfügbar sind. In welcher Reihenfolge werden die Tasks gemäß der Werte aus Bild 8.1 ausgeführt?

T2, T3

Welche Art der Verklemmung tritt auf?

Globale Verklemmung / Deadlock

Die Aussage „Wird die Ausführung von Task 4 erzwungen, nimmt die Semaphore S4 den Wert -2 an.“ ist

[ ] wahr [X] falsch

b) Für die Task *Datenlogging* ist folgender Verlauf gegeben (Bild 8.2). Bestimmen Sie die maximale Antwortzeitdauer und die maximale Ausführzeitdauer.

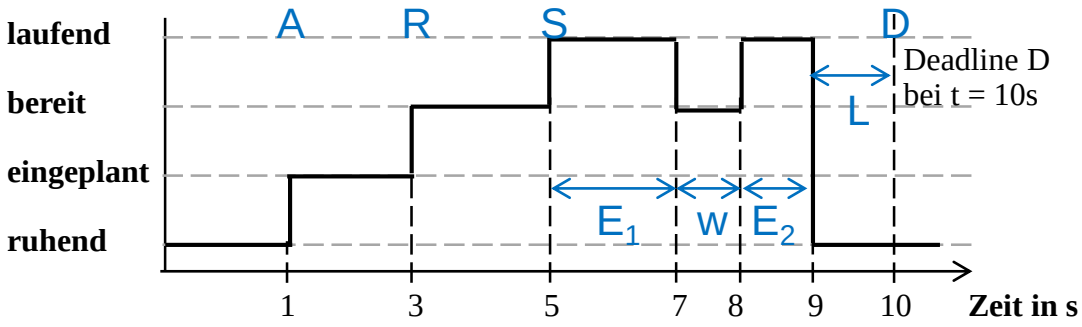


Bild 8.2: *Verlauf Task Datenlogging*

Maximale Antwortzeitdauer (Rechenweg und Ergebnis):

$P = |D - R| = 10 - 3 = 7$ *$P = |R-D|$

Maximale Ausführzeitdauer (Rechenweg und Ergebnis):

$E = E_1 + E_2 = 2 + 1 = 3$

Was muss für den Wert des Spielraums allgemein gelten, um eine Deadline einzuhalten?

Spielraum L muss positiv d.h. >= 0 sein.

9. Echtzeitprogrammiersprache PEARL

a) Die Task *Datenlogging* der Geschwindigkeitsüberwachung soll mit PEARL programmiert werden. Definieren Sie die Task *Datenlogging* mit Priorität 2. Die Task greift über die Semaphor-Variable *Logfile* gesichert auf die gleichnamige Datei zu.

```
// Definitere Task Datenlogging mit Priorität 2

Datenlogging : TASK Priority 2 ;

   // Semaphor-Variable Logfile wird angefragt

      REQUEST Logfile ;
   //  Programmablauf der Task – hier zu vernachlässigen

   // Logfile wird freigegeben

      RELEASE Logfile ;

END ;
```

b) Die Task *Datenlogging* wird um 22:00 Uhr abends beendet (Zustand „ruhend“). Geben Sie die entsprechende Codezeile in PEARL an.

```
AT 22:00:00 TERMINATE Datenlogging;
```

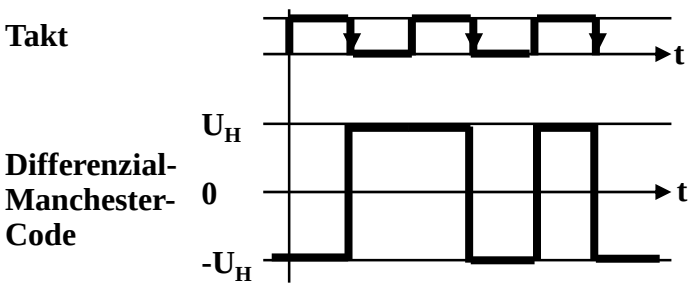
10. Bussysteme und Prozessperipherie

Beantworten Sie die nachfolgenden Fragen.

Datenlogger und Blitzer haben jeweils ihre eigene Steuerung und kommunizieren zusammen auf einem Bus. Nennen Sie eine Buszugriffsart, um bei gleichzeitiger Übertragung den Verlust von Daten zu vermeiden.

CSMA/CA (Carrier Sense Multiple Access /Collision Avoidance) oder TDMA (Time division multiple access)

Sie erhalten das folgende Signal im Differential Manchester-Code. Welche binären Daten (Folge von 1er und 0er) wurden gesendet?



Zeichenfolge der binären Daten:

110

Aufgabe 11: Funktionsbausteinsprache der IEC 61131-3

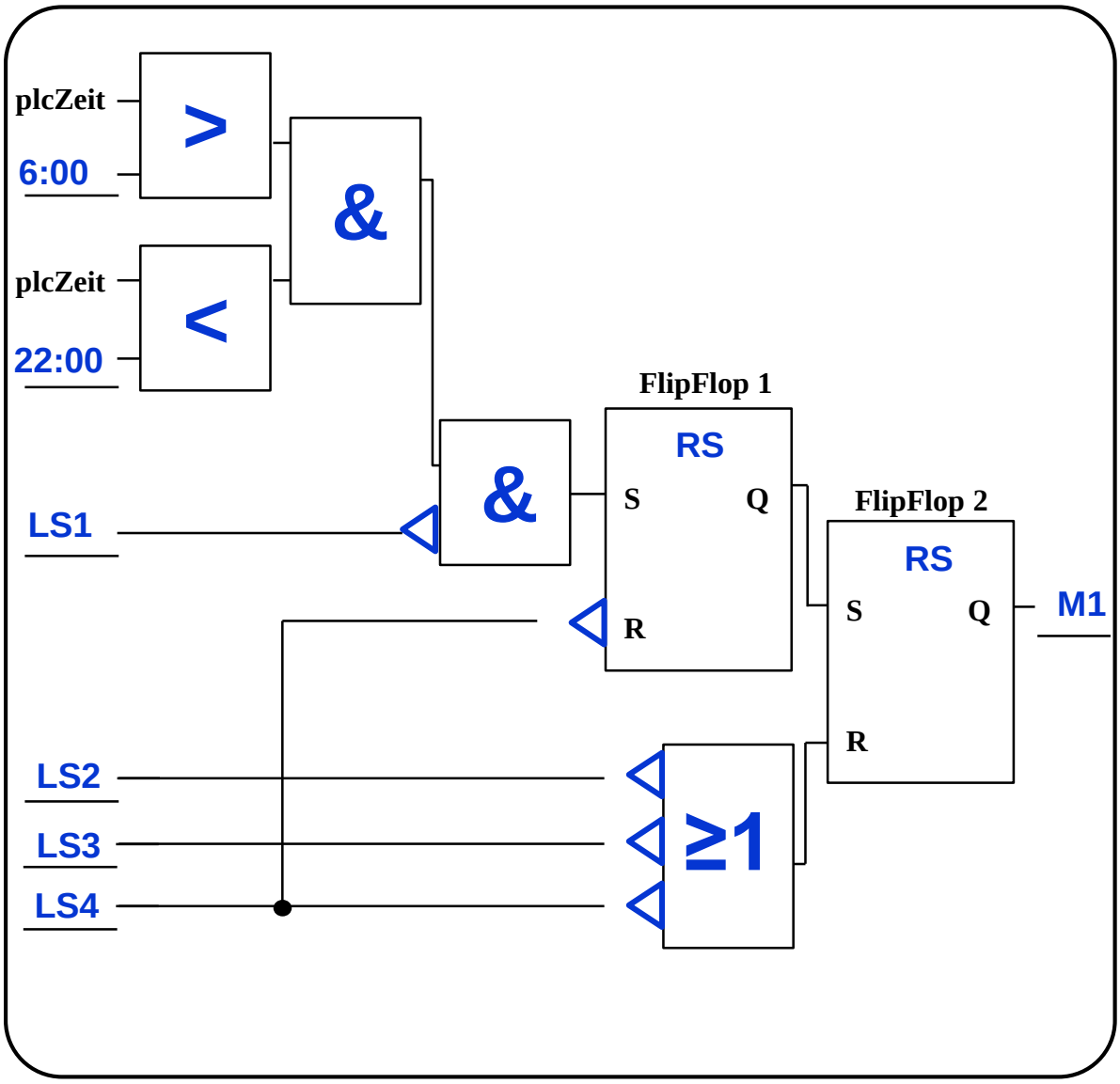
Die Geschwindigkeit eines Autos wird von der Geschwindigkeitsüberwachung mithilfe von vier Lichtschranken (LS) erfasst.

- Wenn zwischen 6:00 Uhr morgens und 22:00 Uhr abends (aktuelle Uhrzeit in Variable plcZeit) ein Auto die LS1 vollständig passiert hat (Lichtschranke sendet wieder „0“), wird die Messung (M1) gestartet (M1 = 1).
- Passiert das Auto LS2 oder LS3 vollständig, wird die Messung kurzzeitig auf 0 gesetzt, um im Anschluss den Geschwindigkeitsverlauf besser nachzuvollziehen.
- Sobald das Auto Lichtschranke 4 (LS4) vollständig passiert hat, wird die Messung komplett gestoppt (und erst wieder bei einem neuen Auto an LS1 aktiviert).

Vervollständigen Sie im Folgenden die (De-)Aktivierung der Messung in IEC 61131-3-Funktionsbausteinsprache (FBS).

Hinweise:

- *Signalverzögerungen im System sind zu vernachlässigen.*
- *Verwenden Sie keine Schaltglieder außer den in der Vorlage bereits vorhandenen.*
- *Ergänzen Sie Negationen falls notwendig.*
- *Statt ≥ bzw. ≤ können Sie auch >= bzw. <= schreiben.*

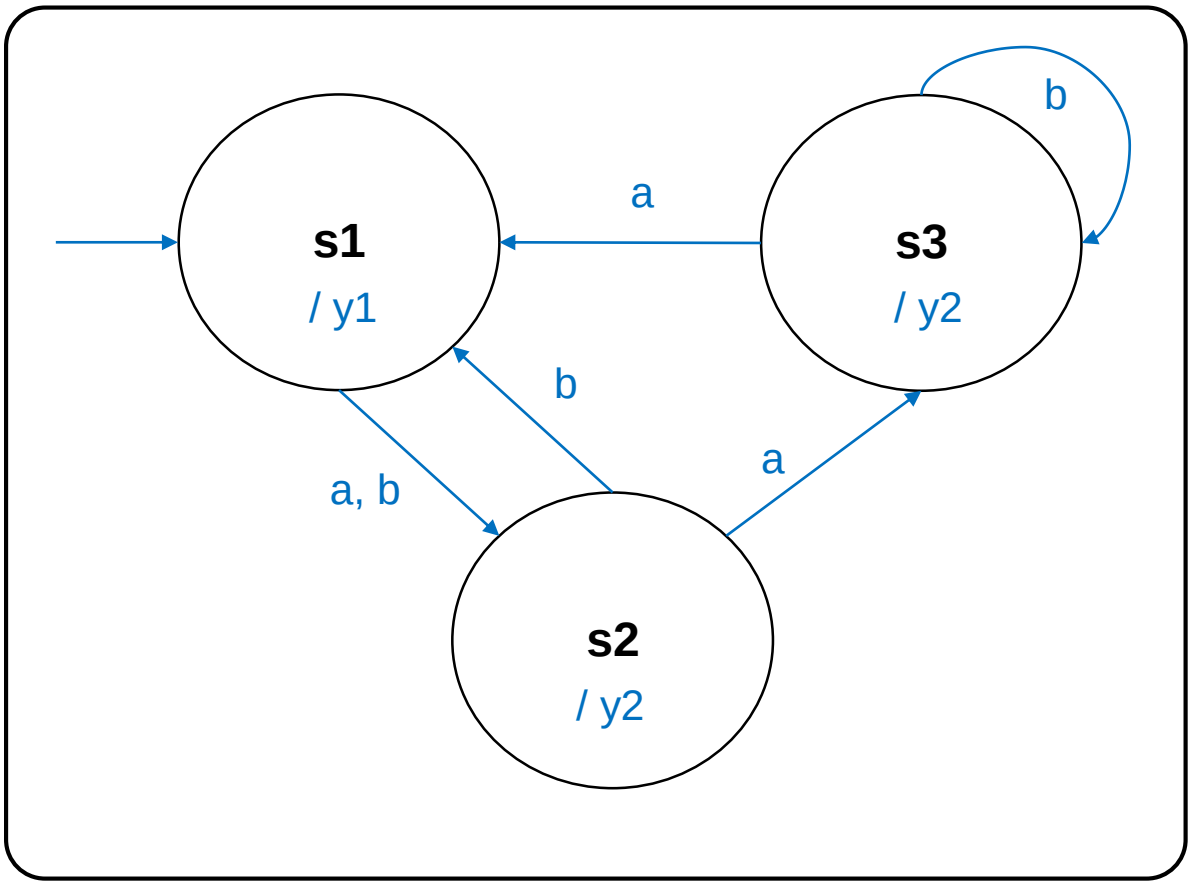


Aufgabe 12: Automaten

Gegeben ist folgende Übergangstabelle für die Zustände s1, s2, s3, den Eingaben a, b und den Ausgaben y1, y2:

T \ S	s1, y1	s2, y2	s3, y2
a	s2	s3	s1
b	s2	s1	s3

a) Zeichnen Sie den entsprechenden Übergangsgraphen. Der Startzustand ist s1.



b) Kreuzen Sie an, welche Ausgabe durch die folgende Eingabe erzeugt wird: b a a b

[X] y1 y2 y2 y1 y2 [ ] y1 y2 y1 y1 y2 [ ] y1 y2 y1 y2 y2

Aufgabe 13: Grundlagen in C

a) Deklarieren Sie ein Array namens `Speicher` vom Datentyp `char` mit Speicherplatz für einen String der Länge drei und belegen Sie es mit dem String `"AAS"` vor. Berücksichtigen Sie hierbei, dass der String einen End-of-String-Character benötigt. Tauschen Sie anschließend den zweiten Wert des Strings durch das Zeichen `'I'` aus und geben Sie den resultierenden String in der Kommandozeile aus.

Hinweis: Ihr Codeausschnitt wird in eine zugrundeliegende Main-Funktion eingebettet, für die die Bibliothek stdio.h bereits eingebunden ist.

```
char Speicher[4] = "AAS";

Speicher[1] = 'I';

printf("%s", Speicher);
```

Alternative Lösungen:

```
char Speicher[4] = {'A','A','S'};
char Speicher[4] = {„AAS“};

Speicher[1] = 'I';

for (int
i=0;i<3;i++){printf("%c",Speicher[
i]);} | puts(Speicher);
```

b) Die folgende Aufgabe betrachtet die Umwandlung von Datentypen. Geben Sie durch Ankreuzen an (nur Einfachantwort möglich), welchen Datentyp die Ergebnisse der angegebenen Berechnungen haben.

`int - float * double`	( )float ( )int ( )char (x)double
`char + int * float`	( )int ( )double (x)float ( )char

c) Programmieren Sie eine Funktion namens `AISDiv`, welche eine Division von zwei Ganzzahlen (`zahlzwei` durch `zahleins`) durchführt. `AISDiv` hat die vorzeichenbehafteten, ganzzahligen 16-bit Übergabeparameter `zahleins` und `zahlzwei` und gibt das Ergebnis als vorzeichenbehaftete 32-bit Gleitkommazahl zurück.

Hinweis: Der Funktion werden nur Wertepaare für zulässige Divisionsoperationen übergeben.

```
float AISDiv (int zahleins, int zahlzwei){

return  (float)zahlzwei/(float)zahleins;

}


Alternative Datentypen für Übergabeparameter:
int16_t oder short
```

d) Bestimmen Sie das Ergebnis des Ausdrucks im Dezimalsystem. Gegeben sind die folgenden Variablen:

```
int a = 8;        int b = 5;          int c = 112;
int *d = &b;      int e = 2;
```

`c - (((a | 2) | *d) >> e)`	**Ergebnis:** 109

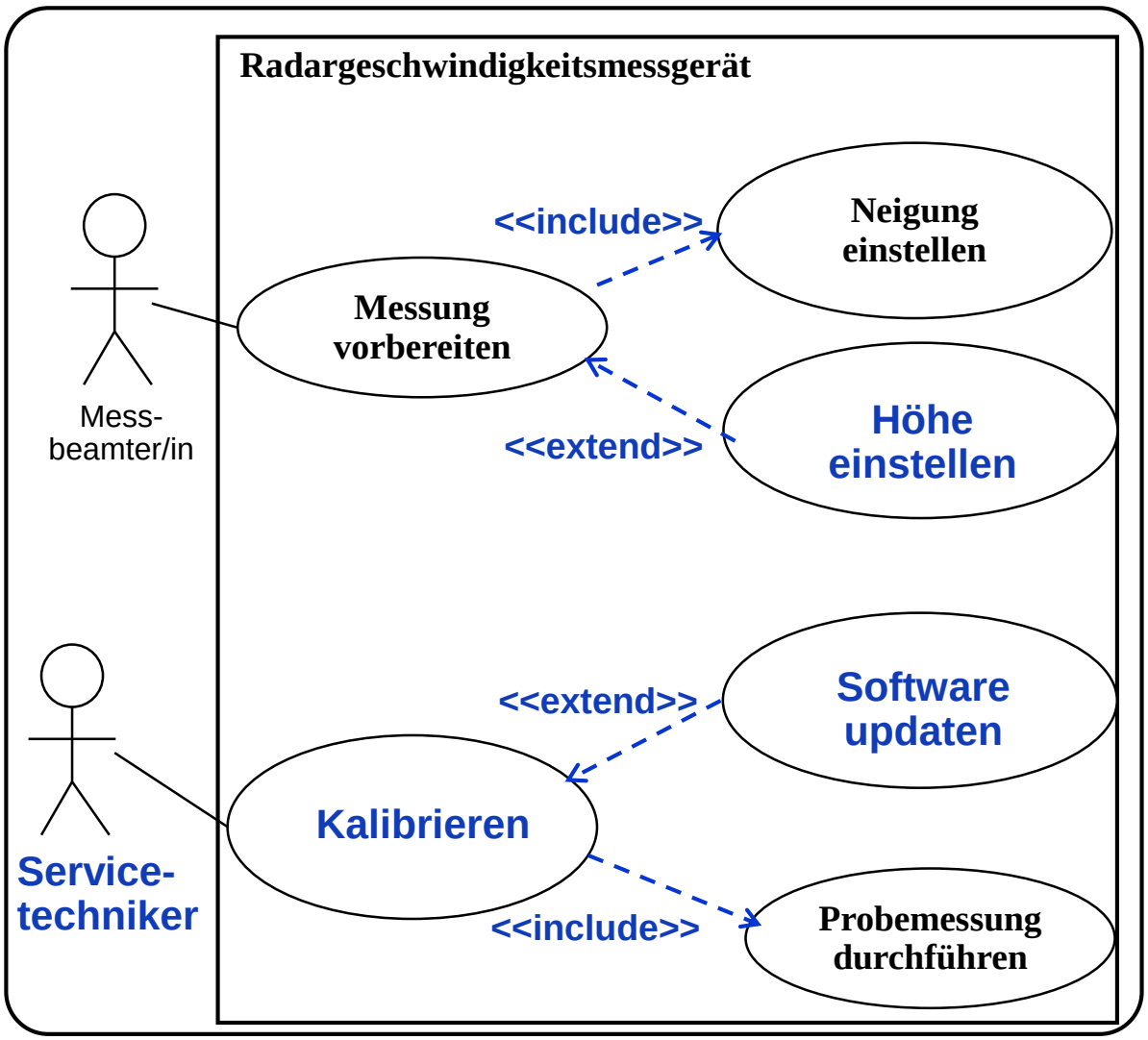
Aufgabe 14: Use-Case-Diagramm

Die oberbayrische Firma AIS Speed Control GmbH hat ein neuartiges Radargeschwindigkeitsmessgerät entwickelt, den SuperBlitzer 3000. In den folgenden Aufgaben soll dieser betrachtet werden.

Das Radargeschwindigkeitsmessgerät wird durch einen/eine Messbeamten/-in bedient. Er/Sie kann die *Messung vorbereiten*. Die Messvorbereitung beinhaltet das Einstellen der Neigung (*Neigung einstellen*). Zusätzlich kann je nach Umgebung die Höhe eingestellt werden (*Höhe einstellen*).

Für eine Zulassung im Straßenverkehr ist es notwendig, dass das Radargeschwindigkeitsmessgerät regelmäßig durch einen/eine *Servicetechniker/in* kalibriert wird. Beim *Kalibrieren* ist es erforderlich, dass eine Probemessung durchgeführt (*Probemessung durchführen*) wird. Falls eine neue Version der Software verfügbar ist, wird diese im Rahmen des Kalibrierens geupdatet. Der Prozess des Kalibrierens kann somit um die Aktion *Software updaten* erweitert werden.

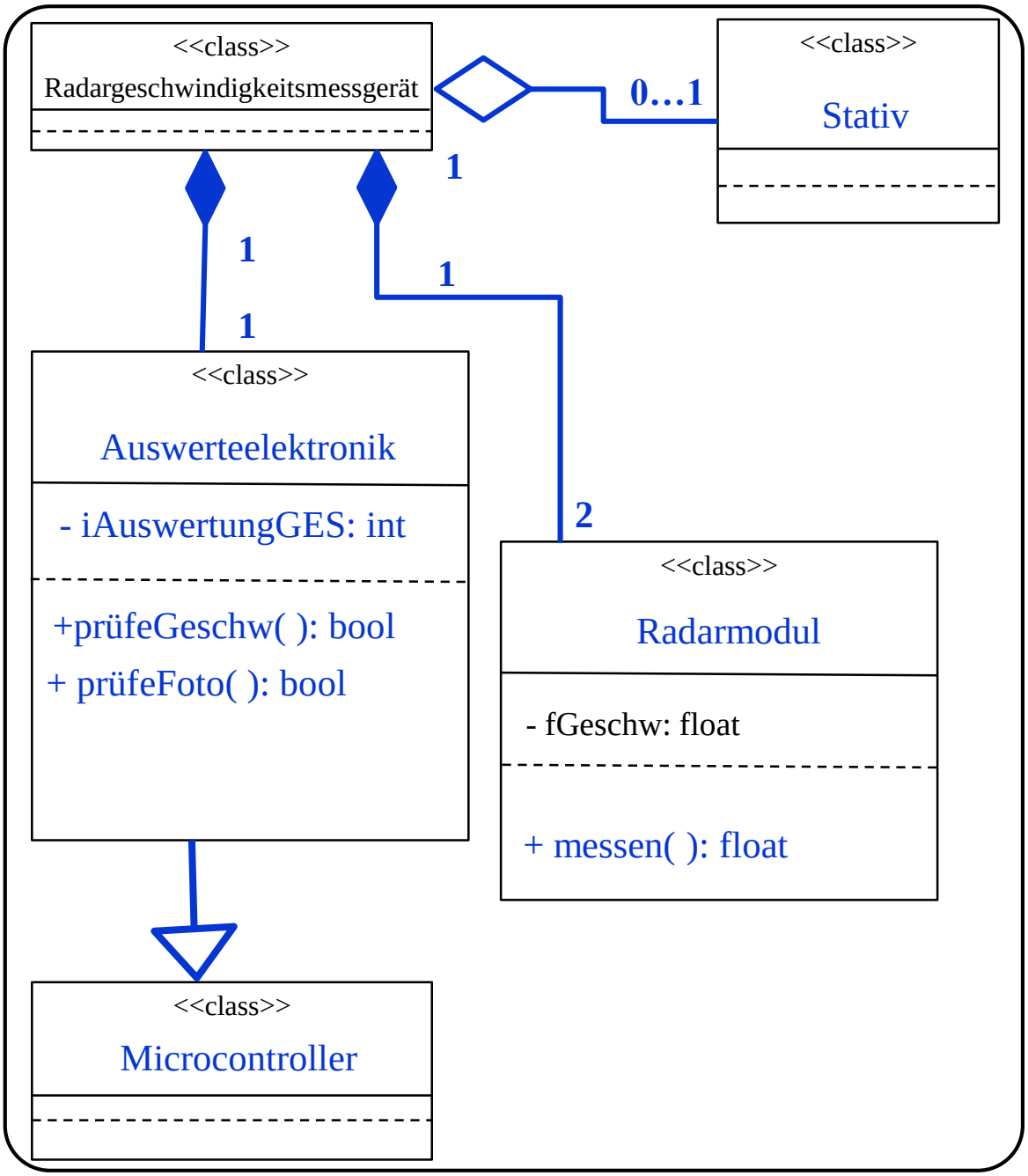
Füllen Sie mithilfe der obigen Angaben das Use-Case-Diagramm der UML für das Radargeschwindigkeitsmessgerät aus. Benennen Sie die Akteure sowie die Anwendungsfälle. Bitte ergänzen Sie bei den Verbindungen zwischen den Use-Cases die richtige Beziehungsart sowie die Richtung (Pfeilspitze).



Aufgabe 15: Klassendiagramm

Das Radargeschwindigkeitsmessgerät besteht immer aus einer *Auswerteelektronik*. Die Auswerteelektronik ist ein *Microcontroller*. Die Auswerteelektronik speichert die Anzahl aller Auswertungen in der Variable *iAuswertungGES* als Ganzzahl. Zudem stellt die Auswerteelekronik die Funktionen *prüfeGeschw* und *prüfeFoto* zur Verfügung. Beide Funktionen liefern das Ergebnis als *bool* zurück. Zusätzlich besteht das Radargeschwindigkeitsmessgerät immer aus genau zwei *Radarmodulen*. Ein Radarmodul stellt die Funktion *messen* bereit. Die Funktion liefert die Geschwindigkeit als float zurück. Das Radarmodul speichert die gemessene Geschwindigkeit in der Variable f*Geschw*. Für den Fall, dass ein Radarmodul ausfällt, kann das Gerät auf das zweite Radarmodul zurückgreifen. Um den Einsatzort noch flexibler wählen zu können, kann das Gerät noch mit einem *Stativ* ausgestattet werden.

Vervollständigen Sie das Klassendiagramm gemäß obiger Beschreibung.



Aufgabe 16: Klassendiagramm zu Code

Sie möchten eine Software zur Verwaltung von Geschwindigkeitsüberwachungsanlagen (kurz: Messanlage) schreiben. Abbildung 16.1 zeigt das vereinfachte UML-Klassendiagramm für die zu implementierende Software.

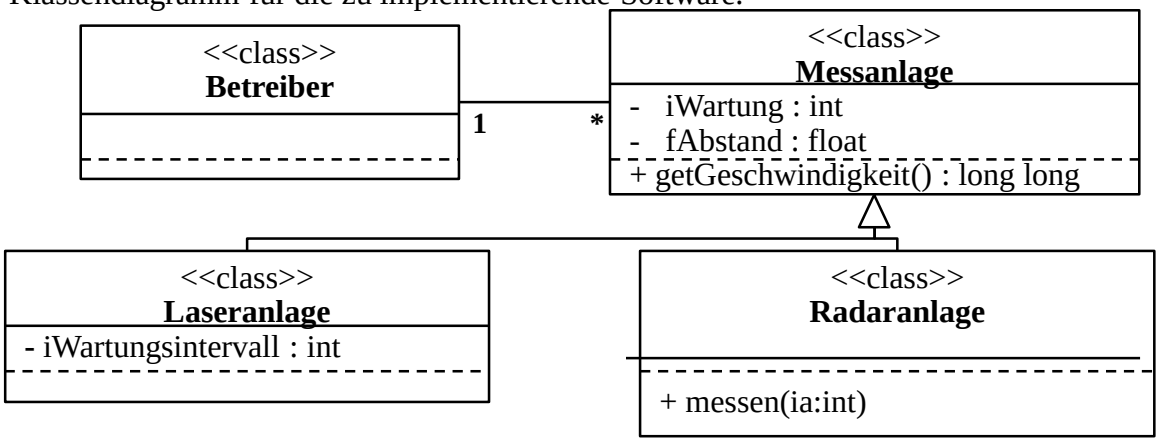


Bild 16.1: *Klassendiagramm*

Implementieren Sie das obige Klassendiagramm in C++, indem Sie das untenstehende Lösungskästchen ergänzen. Deklarieren Sie Attribute und Methoden für die header-Datei. Achten Sie auf die korrekten Sichtbarkeiten, die Parameter sowie die Beziehungen zwischen den Klassen.

```
class Messanlage {
 private:
   int iWartung;
   float fAbstand;
   Betreiber* betreiber;
public:
   long long getGeschwindigkeit();
};

class Laseranlage : public Messanlage {
private:
   int iWartungsintervall;


};

class Radaranlage : public Messanlage {
public:
   void messen(int ia);

};

class Betreiber {};
```

Aufgabe 17: Zustandsdiagramm

Im Folgenden wird eine stationäre Geschwindigkeitsüberwachungsanlage zur Kontrolle der Einhaltung der geltenden Verkehrsregeln betrachtet (Bild 17.1), welche sich in einem Bereitschaftsmodus (`Idle`) befindet. Sofern ein Auto an der Anlage vorbei fährt, sendet diese zunächst ein Radarsignal **(`bRadar = 1` für an, `bRadar = 0` für aus) eine Millisekunde lang** aus. Nach dem Aussenden wird ebenfalls für eine Millisekunde der Messprozess mittels der Funktion **`Messen()`** ausgeführt, um den Abstand des Autos zur Messanlage festzustellen. Dieser Vorgang, bestehend aus dem Aussenden und dem Messen, wird für die Zeitdauer von einer Sekunde wiederholt, sodass anschließend in der Funktion `Auswerten()` die Geschwindigkeit des Autos durch seine Positionsänderung abgeleitet werden kann. Sofern die Geschwindigkeit des Autos kleiner gleich 80 ist, werden die Messdaten durch die Funktion `Datenloeschen()` verworfen. Andernfalls werden von der Geschwindigkeitsüberwachungsanlage ein Blitz durch die Funktion `Blitzen()` ausgelöst und ein Bild des Autos mittels der Funktion `BildSpeichern()` aufgezeichnet und gespeichert. Mit dem Löschen der Daten oder dem Speichern des Bildes ist der Geschwindigkeitsüberwachungsvorgang für ein Auto beendet und die Anlage wird in den `Idle` Zustand versetzt, bis das nächste Auto vorbeifährt.

Zusatzinfo: Die Variable `timer` zählt in Millisekunden hoch und kann gesetzt werden, woraufhin diese erneut zu zählen beginnt. Nach jedem Setzvorgang wird der Timer wieder auf 0 gesetzt.

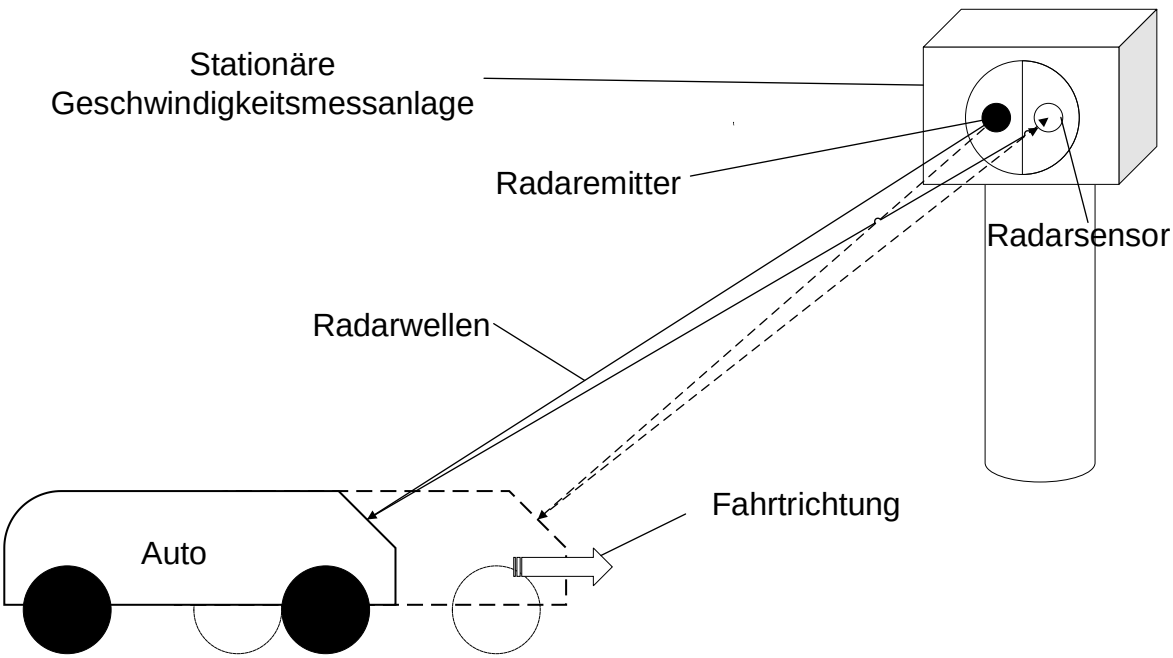


Bild 17.1: *Skizze der stationären Geschwindigkeitsüberwachungsanlage*

Es ist das in Bild 17.2 gezeigte Zustandsdiagramm mit den Zustandsnummern ① bis ⑥ gegeben. Füllen Sie die durch römische Ziffern gekennzeichneten Lücken im Lösungsfeld aus, bzw. beantworten Sie die angegebenen Fragen.

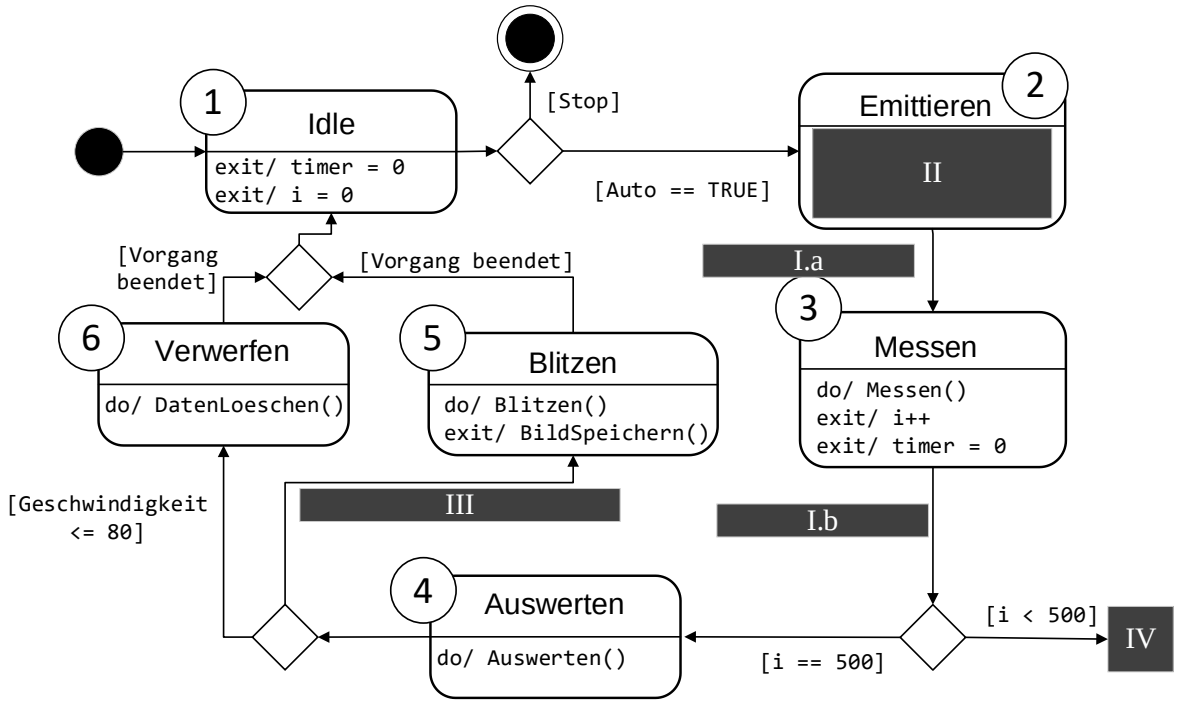


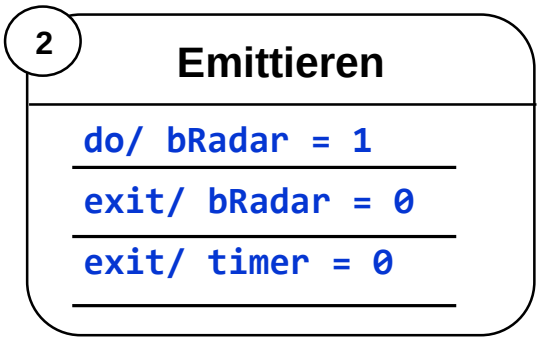
Bild 17.2: *Unvollständiges Zustandsdiagramm des Betriebsablaufs*

I. Geben Sie die korrekte Wächterbedingungen an:

I a. **[timer == 1ms]**

I b. **[timer == 1ms]**

II. Modellieren Sie den Zustand `Emittieren` korrekt aus:



III. Geben Sie die korrekte Wächterbedingung an:

[Geschwindigkeit > 80]

IV. Welcher Zustand muss nach dem Messen folgen (nach Zustand ③), um den Ablauf wie in der Beschreibung zu gewährleisten:

2

Aufgabe 18: Zustandsdiagramm zu C-Code

Sie haben nun die Aufgabe, Teile des in Aufgabe 17 modellierten Zustandsdiagramms in Form von C-Code zu implementieren. Hierfür stehen Ihnen die in Tabelle 18.1 dargestellten und bereits implementierten Funktionen zur Interaktion mit der stationären Geschwindigkeitsüberwachungslage, die Ein- und Ausgänge sowie erweiterten Variablen zur Verfügung.

Tabelle 18.1*: *Sensor- und Aktorvariablen der Messanlage, sowie erweiterte Variablen und bereits implementierte Funktionen

Typ	Name	Beschreibung
FUNKTIONEN	`Messen()`	Misst die Zeitspanne zwischen Aussenden des Radarsignals und Eintreffen des Radarsignals.
	`Auswerten()`	Berechnet die Geschwindigkeit basierend auf den aus Messen() erhaltenen Daten.
	`DatenLoeschen()`	Löscht die in Auswerten() errechneten Daten.
	`Blitzen()`	Aktiviert den Blitz und nimmt ein Foto auf.
	`BildSpeichern()`	Speichert das durch die Funktion Blitzen() erzeugte Bild in einer Datenbank ab.
	`lSensoren (SENSOREN *s)`	Einlesen der aktuellen Sensorwerte vom Typ SENSOREN.
	`sAktoren (AKTOREN *a)`	Übertragung der aktuellen Aktorwerte von Typ AKTOREN zum Schreiben auf die Hardware.
SENSOREN / AKTOREN	`s.Radarsensor`	Empfängt Radarwellen.
	`s.Kamera`	Nimmt Foto von Nummernschild auf.
	`a.Radaremitter`	Sendet Radarsignal aus.
	`a.Blitz`	Beleuchtet das Nummernschild für gute Lichtverhältnisse beim Foto.
VARIABLEN	`vplcZeit`	Aktuelle Laufzeit des Programms in Millisekunden (positive Ganzzahl).
	`t`	Variable für timer-Programmierung (positive Ganzzahl, zählt in Millisekunden).
	`zustand`	Aktueller Zustand des Zustandsautomaten (Ganzzahl), welcher ausgeführt wird.
	`Geschwindigkeit`	In der Funktion Auswerten() ermittelte Geschwindigkeit des Autos.
	`*pCPUZeit`	Pointer, der für die Timerfunktion verwendet wird und deshalb auf vplcZeit zeigen soll.

a) Programmgrundgerüst

Vervollständigen Sie das im folgenden Lösungskästchen gezeigte Programmgerüst, um eine zyklische Ausführung des Zustandsautomaten zu ermöglichen. Verwenden Sie hierfür die in Tabelle 18.1 angegebenen Variablennamen. Zur Interaktion mit der Hardware verwenden Sie die ebenfalls in Tabelle 18.1 gegebenen Funktionen, welche im Header `Geschwindigkeitsmessung.h` bereits definiert und implementiert sind. Der Platzhalter `/* ZUSTAENDE */` soll später durch den spezifischen Code der Zustände in Form eines Zustandsautomaten ersetzt werden.

```
// Header einbinden
#include "Geschwindigkeitsmessung.h"

SENSOREN s;          // Sensorik vom Typ SENSOREN
AKTOREN a;           // Aktorik vom Typ AKTOREN
UHR vplcZeit;

int main()
{  // Deklarationen
   unsigned int* pCPUZeit;          // Zeitvariable
   int zustand;                     // Schrittvariable

   while(1)                        // Zyklische Ausfuehrung
   {
         // Adresse für Timer uebergeben
         pCPUZeit = &vplcZeit;
         // Einlesen von Hardware und Zeit
         lSensoren(&s);;
         switch(zustand);
         // Zustandsautomat
         {
                  /* ZUSTAENDE */
         }
          // Schreiben auf Hardware
         sAktoren(&a);
   }
   return 0;      // Wird nicht erreicht
}
```

b) Implementierung des Zustands „`Auswerten`“

Der Zustand „`Auswerten`“ wurde im Folgenden um eine Wartezeit erweitert und soll nun implementiert werden. Der Rumpf des Falls (`case 4`) ist bereits vorgegeben. Ergänzen Sie den Code, sodass der Zustand, wie im Zustandsdiagramm (Bild 18.1), korrekt umgesetzt wird. Verwenden Sie hierfür die Variablen, sowie die Ein- und Ausgänge aus Tabelle 18.1. Beachten Sie, dass mehr Leerzeilen zur Verfügung stehen, als für die korrekte Antwort erforderlich sind.

Achtung: Die Funktion `timer()` ist Ihnen hier nicht gegeben, deren Funktionalität soll aber in Case 4 realisiert werden. Die Variable **t** ist bei Eintritt in einen Zustand immer 0. Dies muss auch für die folgenden Zustände gewährleistet werden. Case 4 ist eingebettet in den Zustandsautomaten aus der Aufgabe a). Die Variable **vplcZeit** zählt kontinuierlich in Millisekunden hoch.

```
case 4:   // Auswerten
Auswerten();

if (t==0)
{
  t = vplcZeit + 5;
}
else if (vplcZeit >= t)
{
  t = 0;
  if (Geschwindigkeit <= 80)
  {
     zustand = 6;
  }
  else if (Geschwindigkeit >80)
  {
     zustand = 5;
  }
}

break;
```

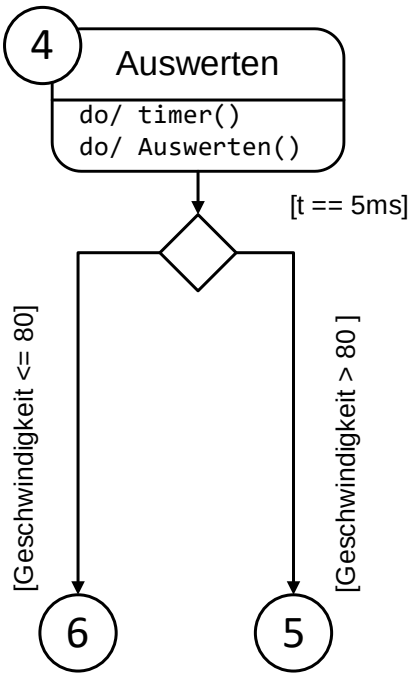


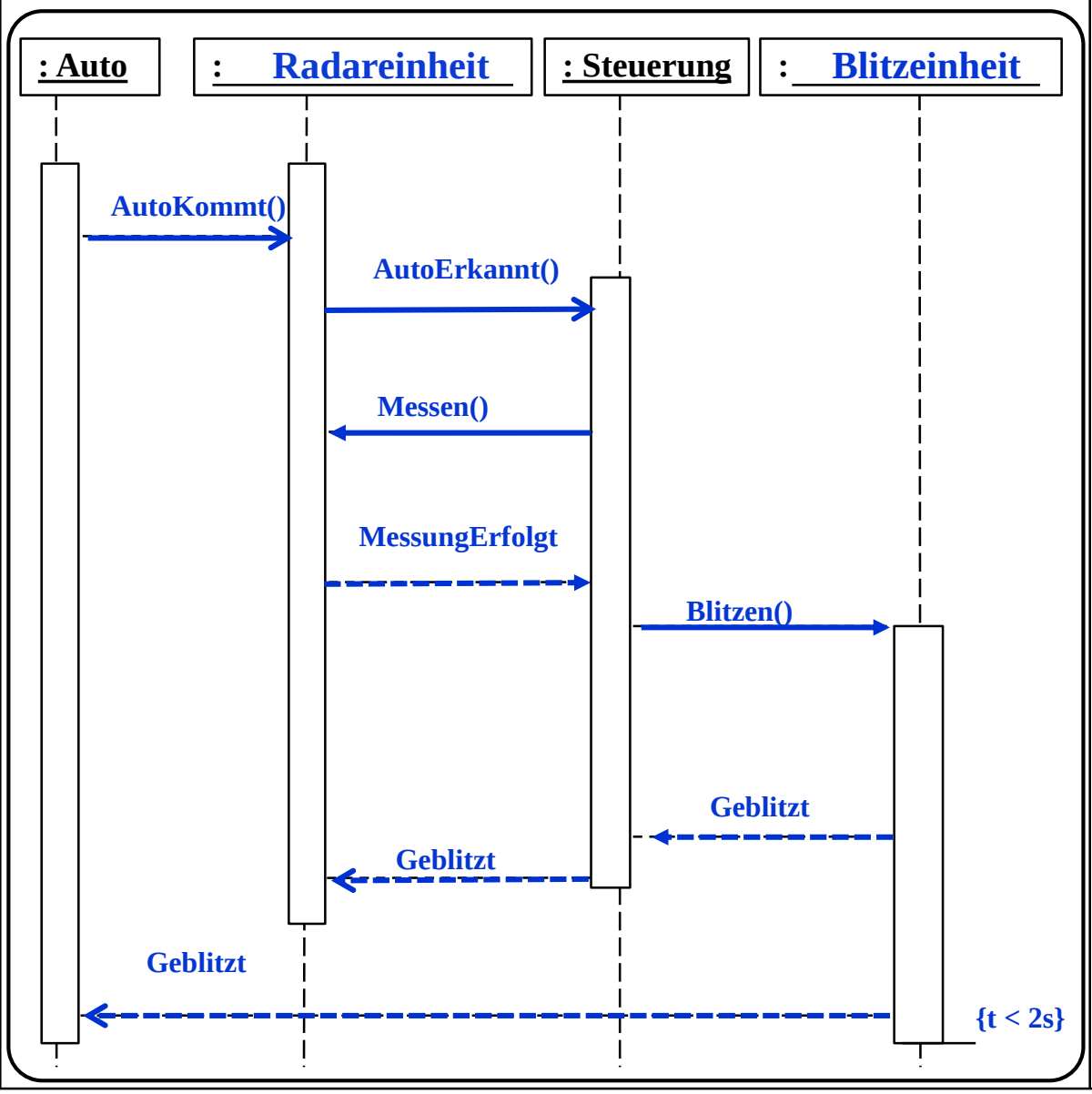
Bild 18.1: *Zustände 4, 5 und 6*

Aufgabe 19: Sequenzdiagramm zum Szenario „*Blitzen*“

Das **Auto** triggert die **Radareinheit**, was hier mit dem Befehl *AutoKommt()* angestoßen wird. Die **Radareinheit** stößt die **Steuerung** mit *AutoErkannt()* an, um die stationäre Geschwindigkeitsmesseinheit in den Messzustand zu versetzen. Die **Steuerung** startet dann die Messung der **Radareinheit** (*Messen()*). Sobald die Messung erfolgt ist, antwortet die **Radareinheit** der **Steuerung** mit *MessungErfolgt*. In unserem Beispiel gehen wir davon aus, dass die Geschwindigkeit von 80km/h überschritten wurde und deshalb ein Blitz ausgelöst wird. Das Blitzen wird durch die **Steuerung** mit dem Kommando *Blitzen()* gestartet und nach Erfolg des Blitzvorgangs sendet die **Blitzeinheit** eine Bestätigung mit *Geblitzt*. Die **Blitzeinheit** soll aufgrund ihres hohen Energieverbrauchs höchstens *2 Sekunden* Aktiv sein. Die Rückmeldung *Geblitzt* wird auch von der **Steuerung** an die **Radareinheit** übergeben. Wir gehen davon aus, dass mittels **Blitzeinheit** die Nachricht *Geblitzt* auch an das **Auto** übergeben wird.

Ergänzen Sie das Sequenzdiagramm entsprechend der Beschreibung.

Alle Nachrichten sind mit gestrichelten Linien vorgegeben. Ergänzen Sie die Pfeilspitzen und ändern Sie – falls notwendig – die Linie in eine durchgezogene Linie.



Aufgabe 20: Algorithmen und Datenstrukturen

Eine stationäre Geschwindigkeitsüberwachungsanlage misst die Geschwindigkeit der vorbeifahrenden Autos über ein fest definiertes Zeitintervall von 1 Sekunden in 1 Millisekunden-Schritten und speichert die Daten zu dem zugehörigen Auto in einer Datenbank. Implementieren Sie eine einfache Datenbankfunktionalität in Form einer **einfach verketteten Liste,** wobei jedes erfasste Auto eineindeutig einem Element der Liste zugeordnet wird. Die Listenelemente werden durch einen Zeiger (`pnext`) verkettet. Der Zeiger (`pfirst`) kennzeichnet das erste Listenelement.

In einem **Listenelement** vom Typ `AUTO` sollen gespeichert werden:

- Gemessene Geschwindigkeiten des Autos als 64-bit Fließkommazahl in einem Array (`Geschwindigkeit`). Wählen Sie eine geeignete Arraylänge unter der Bedingung, dass Sie den Speicher effizient nutzen.
- Maximale Geschwindigkeit des Autos (`MaxGeschwindigkeit`) als 64-bit Fließkommazahl mit 2 Nachkommastellen.
- Binärer Kennzeichner (`Flag`) als vorzeichenlose 8-bit Ganzzahl, welcher angibt, ob die Messung erfolgreich (`Flag=1`) oder fehlerhaft (`Flag=0`) ist.
- Kennzeichen des Autos (`Kennzeichen`) mit 8 Ziffern oder Buchstaben als String. Berücksichtigen Sie hierbei, dass der String einen End-of-String-Character benötigt.

Im **Listenkopf** vom Typ `BLITZER` sollen gespeichert werden.

- Gesamtanzahl der erfassten Autos (`Anzahl`) als vorzeichenlose 32-bit Ganzzahl.

a) Vervollständigen Sie die nachfolgenden Lösungskästchen zur Definition des Listenkopfes und der Listenelemente gemäß der obigen Beschreibung.

```
typedef struct ____ AUTO{
    struct AUTO* pnext;
    double Geschwindigkeit[1000];
    double MaxGeschwindigkeit;
    unsigned char Flag;
    char Kennzeichen[9];
    ____
} AUTO;
```

```
typedef struct ____ {
    AUTO* pfirst;
    unsigned int Anzahl;
    ____
} BLITZER;
```

Um Fehler bei der Geschwindigkeitsmessung zu erkennen, werden die gemessenen Geschwindigkeiten in dem Array (`Geschwindigkeiten`) über die Messdauer integriert, um die daraus berechnete Strecke mit einem vorgegebenen Sollwert zu vergleichen.

b) Implementieren Sie eine Funktion `integrieren`, welche das Integral zwischen den Stützstellen k und m mit der Trapezregel berechnet (Bild 20). Dabei wird der Flächeninhalt unter dem Integral durch Trapeze genähert. Die Funktion `integrieren` soll als **Parameter** einen Pointer auf das betreffende Listenelement sowie die Indices m und k des Geschwindigkeitsarrays übergeben bekommen. Der Rückgabewert ist der Wert des Integrals vom Typ double. Der Stützstellen-abstand n beträgt gemäß Aufgabe 0.001.

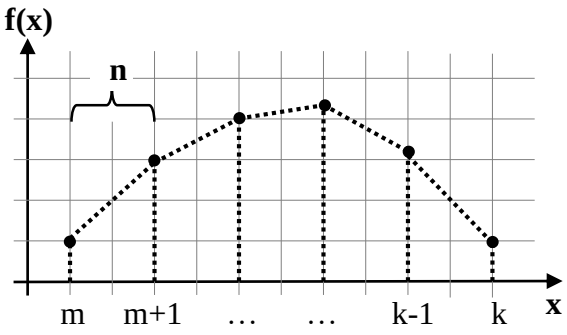


***Bild 20**: Prinzipskizze zur Trapezregel, Messpunkte (•),*

Trapezregel: $\mathbf{Integral_{[k,m]} = \frac{n}{2} \cdot (f(m) + 2 \cdot f(m+1) + .. + 2 \cdot f(k-1) + f(k))}$

```
double     integrieren (AUTO  *data,int  m ,int  k )
{
   //Wert des Integrals, Hilfspointer
   double integral, *geschw;

   //Zugriff auf das Geschwindigkeitsarray
   geschw =          data->Geschwindigkeit;

   //Stuetzstellenabstand
   float n = 0.001;

   //Berechnen des Integrals mit der Trapezregel
      integral = *(geschw + k) + *(geschw + m);

   for(  int i=m+1    ;   i<=k-1              ;i++)
   {
         integral += 2 · *(geschw + i);
   }

   integral = integral · n / 2;

   return integral;
}
```

In unregelmäßigen Abständen soll die Anzahl der gemessenen Autos (Anzahl der Elemente in der Liste) ausgezählt werden und das Ergebnis im Listenkopf unter der Variable (`Anzahl`) gespeichert werden.

c) Implementieren Sie eine Funktion `statistik`, welche die Anzahl an gemessenen Autos, also die Anzahl an Listenelementen, berechnet und das Ergebnis im Listenkopf speichert. Überlegen Sie sich dazu, welchen Typ von Funktionen Sie benötigen, um den Wert der Variablen (`Anzahl`) im Listenkopf zu verändern. Die Funktion `statistik` soll als Übergabeparameter einen Zeiger auf den Listenkopf erhalten.

```
void        statistik (_BLITZER__ *data)
{
    //Zaehlvariable für Anzahl an Listenelementen
    unsigned int i = 0;

    //Pointer auf erstes Listenelement
    _AUTO*__ temp = _data->pfirst;_____________

    //Listenelemente durchzaehlen
    _while (temp->pnext != NULL)_____________
    _{_____________
          temp = temp->pnext;_____________
          i++;_____________
     }

    //Anzahl im Listenkopf aktualisieren
    _data->Anzahl = i;_____________
    _____________
}
```

Alle Daten in der einfach verketteten Liste wurden nun mittels vordefinierter Funktionen gemessen oder berechnet, sodass in jedem Listenelement `Geschwindigkeit`, `MaxGeschwindigkeit`, `Flag` und `Kennzeichen` mit den zugehörigen Werten initialisiert sind.

d) Implementieren Sie eine Funktion `schreiben`, welche für ein Listenelement das Kennzeichen (`Kennzeichen`) und die maximale Geschwindigkeit (`MaxGeschwindigkeit`) in eine Datei `daten.txt` hineinschreibt. Die Funktion soll als Übergabeparameter einen Pointer auf das betreffende Listenelement übergeben bekommen. Die Datei `daten.txt` soll genau dann zum Schreiben geöffnet werden, wenn die Messung erfolgreich war (`Flag=1`).

Hinweis: Wie in Aufgabenteil a) erwähnt, soll die maximale Geschwindigkeit des Autos (`MaxGeschwindigkeit`) als 64-bit Fließkommazahl mit 2 Nachkommastellen ausgegeben werden. Die Daten sollen nach folgendem Schema geschrieben werden:

M-NR-120;80.05

Beispiel für data.txt

```
void schreiben (AUTO *data)
{
   if(data->Flag == 1)
   {
     FILE* p = fopen("daten.txt","w");
     fprintf(p,"%s;%.2f",data->Kennzeichen,
                    data->MaxGeschwindigkeit);

     fclose(p);
   }
}
```