Lehrstuhl für Automatisierung und Informationssysteme
Prof. Dr.-Ing. B. Vogel-Heuser

Vorname:	
Nachname:	
Matrikelnummer:	

Prüfung – Informationstechnik

Sommersemester 2014

22.08.2014

Bitte legen Sie Ihren Lichtbildausweis bereit.

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält 30 nummerierte Seiten inkl. Deckblatt.
Bitte prüfen Sie die Vollständigkeit Ihres Exemplars!

Bitte nicht mit rot oder grün schreibenden Stiften oder Bleistift ausfüllen!

Diesen Teil nicht ausfüllen.

Aufgabe	GL	BS	MSE	C		∑	**Note**
erreichte Punkte							
erzielbare Punkte	48	48	48	96		240	

Nachname, Vorname _____ Matrikelnummer _____

Aufgabe GL: Zahlensysteme und Logische Schaltungen

Aufgabe GL:
48 Punkte

1. Zahlensysteme

a) Überführen Sie die unten angegebenen Zahlen in die jeweils anderen Zahlensysteme. **Hinweis***:* Achten Sie genau auf die jeweils angegebene *Basis*!

1 (20)$_7$ = ( )$_{16}$ = ( )$_8$

2 (0100 1100,11)$_2$ = ( )$_{10}$

b) Rechnen Sie die unten angegebene Zahl (angelehnt an IEEE 754) in eine Dezimalzahl um. Die Größe der Speicherbereiche für Vorzeichen, Mantisse und Exponent entnehmen Sie der folgenden Speicherdarstellung.

1	1	0	1	0	0	1	0	0	0	1	1	1	0
V	e (5 Bit)					M (8 Bit)							

Hinweis: Ergebnisse und Nebenrechnungen außerhalb der dafür vorgesehenen Textblöcke werden nicht bewertet!

Vorzeichen: V (+ oder -)

V =

Biased Exponent: e (in Dezimal)

e =

Bias: B; Exponent: E (in Dezimal)

B = **E =**

Mantisse: M (mit Vorkommastelle)

M =

Mantisse: M (denormalisiert)

M =

Dezimalzahl

Z =

Nachname, Vorname _____ Matrikelnummer _____

2. Boolesche Algebra / Logische Schaltungen

Gegeben ist die folgende Normalform mit drei Eingängen und einem Ausgang.

$$y = (\bar{a} \vee b \vee c) \wedge (\bar{a} \vee \bar{b} \vee c) \wedge (a \vee \bar{b} \vee c) \wedge (a \vee b \vee \bar{c})$$

a) Füllen Sie die Wahrheitstabelle für das Ausgangssignal y bei gegebenen Eingangsbelegungen der Variablen a, b und c aus.

Dez	a	b	c	y
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

b) Welcher Normalform entspricht die oben angegebene Schaltung?

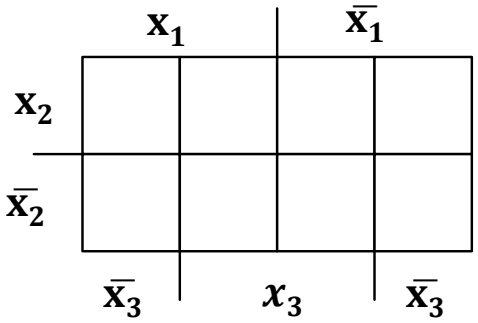
Nachname, Vorname _____ Matrikelnummer _____

3. Normalformen und Minimierung

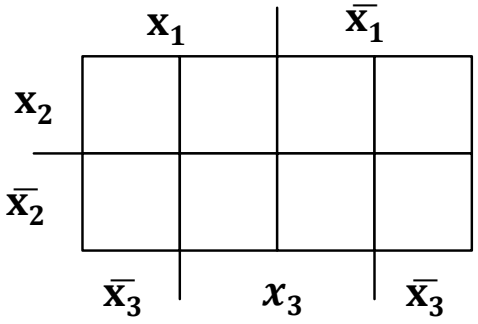
Gegeben ist die folgende Wahrheitstabelle.

Dez	x_1	x_2	x_3	y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

a) Tragen Sie die Ausgangsvariable y aus der Wahrheitstabelle in das KV-Diagramm ein und minimieren Sie die KNF (*Konjunktive Normalform*) mit Hilfe des KV-Diagramms. *Schreiben Sie ebenfalls die minimierte Funktion in boolescher Algebra auf.*



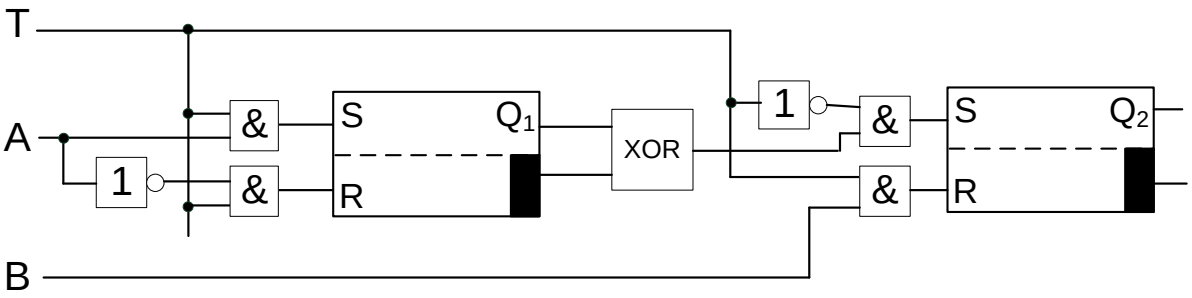
b) Tragen Sie die Ausgangsvariable y aus der Wahrheitstabelle in das KV-Diagramm ein und minimieren Sie die DNF (*Disjunktive Normalform*) mit Hilfe des KV-Diagramms. *Schreiben Sie ebenfalls die minimierte Funktion in boolescher Algebra auf.*



Nachname, Vorname _____ Matrikelnummer _____

4. Flip-Flops

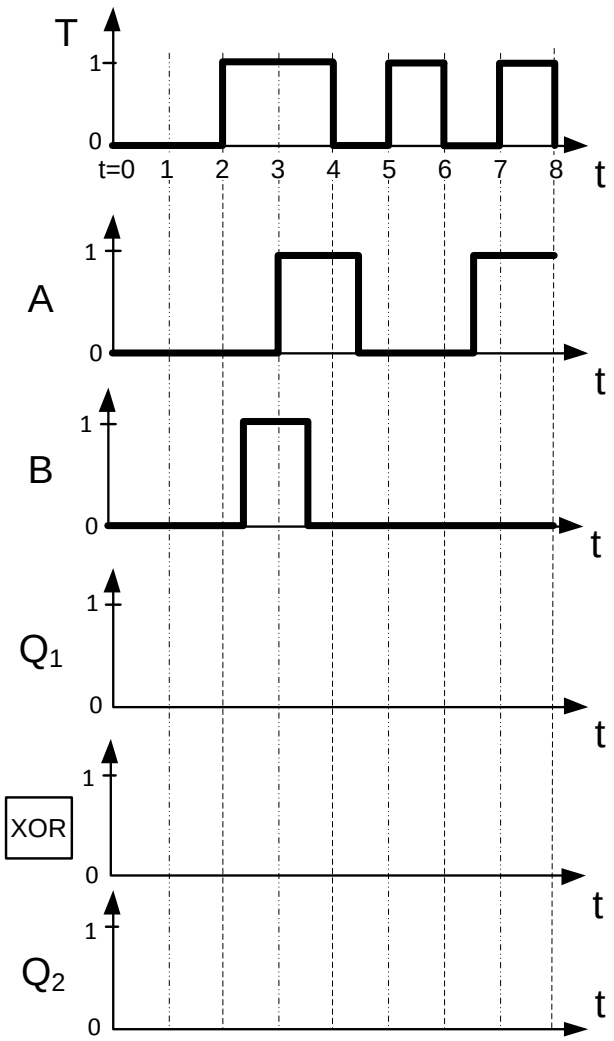
Gegeben ist die folgende Master-Slave-Flip-Flop-Schaltung.



Bei t = 0 sind die Flip-Flops in folgendem Zustand: $Q_1 = 0$ und $Q_2 = 1$

Analysieren Sie die Schaltung, indem Sie für die Eingangssignale A, B und T die zeitlichen Verläufe für Q_1, Q_2 und XOR in die vorgegebenen Koordinatensysteme eintragen.

Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden.



Nachname, Vorname

Matrikelnummer

5. Befehlsabarbeitung mit MMIX-Rechnerarchitektur

Im Registerspeicher eines MMIX-Rechners befinden sich die in Tabelle GL-5.3 (siehe folgende Seite) gegebenen Werte. Es sollen nacheinander die vier Befehle (Tabelle GL-5.5) abgearbeitet und das Ergebnis in dem Datenspeicher (Tabelle GL-5.4) abgelegt werden.

Ergänzen Sie zunächst die Tabelle GL-5.5, indem Sie die gegebenen Maschinen- oder Assemblerbefehle bzw. Befehlsbeschreibung in die jeweils fehlende Form umwandeln (ein Beispiel finden Sie in Tabelle GL-5.2). Führen Sie dann diese Befehle mit den Werten des Registerspeichers durch (Tabelle GL-5.3, Spalte „Wert vor Befehlsausführung“) füllen Sie den Registerspeicher nach der Befehlsausführung vollständig aus (Tabelle GL-5.3, Spalte „Wert nach Befehlsausführung“) und tragen Sie in den Datenspeicher (Tabelle GL-5.4) nur die aus der Bearbeitung der Befehle resultierenden Werte ein.

	0x_0	0x_1	…	0x_4	0x_5	…
	0x_8	0x_9		0x_C	0x_D	
0x0_	TRAP	FCMP	…	FADD	FIX	…
	FLOT	FLOT I		SFLOT	SFLOT I	
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	
0xF_	JMP	JMP B	…	GETA	GETA B	…
	POP	RESUME		SYNC	SWYM	

Tabelle GL-5.1: MMIX-Code-Tabelle

Maschinensprache	Assemblersprache	Befehlsbeschreibung
z. B. 0x8D 9B A1 24	*LDO $0x9B, $0xA1, $0x24*	*M[$0xA1 + $0x24] = $0x9B oder: „Lädt den Wert der Datenspeicherzelle an der Adresse A1 plus Offset 24 als Octa in die Registerzelle 9B.“*

Tabelle GL-5.2: Lösungsbeispiel

Nachname, Vorname _____ Matrikelnummer _____

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Wert nach Befehlsausführung
…	…	…
$0x9D	0x00 00 00 00 00 00 00 A9	0x
$0x9E	0x00 00 00 00 00 00 01 01	0x
$0x9F	0x00 00 00 00 0B 12 11 05	0x
$0xA0	0x00 00 00 00 00 00 61 00	0x
…	…	…

Tabelle GL-5.3: Registerspeicher

Datenspeicher																			
Wert	…																		…
Adresse	…	**0x0..0 61 FA**	**0x0..0 61 FB**	**0x0..0 61 FC**	**0x0..0 61 FD**	**0x0..0 61 FE**	**0x0..0 61 FF**	**0x0..0 62 00**	**0x0..0 62 01**	**0x0..0 62 02**	**0x0..0 62 03**	**0x0..0 62 04**	**0x0..0 62 05**	**0x0..0 62 06**	**0x0..0 62 07**	**0x0..0 62 08**	**0x0..0 62 09**	**0x0..0 62 0A**	**0x0..0 62 0B**	…

Tabelle GL-5.4: Datenspeicher

Maschinensprache	Assemblersprache	Befehlsbeschreibung
0x18 9D 9D 9E		
0xA4 9F 9E A0		
	INCH $0xA0, 0x00 A0	
		$0x9E = $0x9F - 0x07

Tabelle GL-5.5: Maschinensprache – Assemblersprache – Befehlsbeschreibung

Nachname, Vorname _____ Matrikelnummer _____

Aufgabe BS: Betriebssysteme

Aufgabe BS:
48 Punkte

1. Scheduling

Sechs Prozesse (P1 bis P6) sollen mit einem Einkernprozessor abgearbeitet werden. Das Diagramm BS-1.1 zeigt die Zeiten, zu denen die Prozesse am Einkernprozessor eintreffen und die Ausführungszeiten der einzelnen Prozesse. Prozess 1 tritt periodisch dreimal auf. Die Prozesse sollen zur Laufzeit mit unterschiedlichen Schedulingverfahren eingeplant werden. Alle Schedulingverfahren beginnen zum Zeitpunkt t = 0T. Für die Schedulingverfahren, bei denen feste Prioritäten berücksichtigen werden müssen, ist in der Tabelle BS-1.2 die entsprechende Prioritätenverteilung (Prioritäten 1 und 2) gegeben.

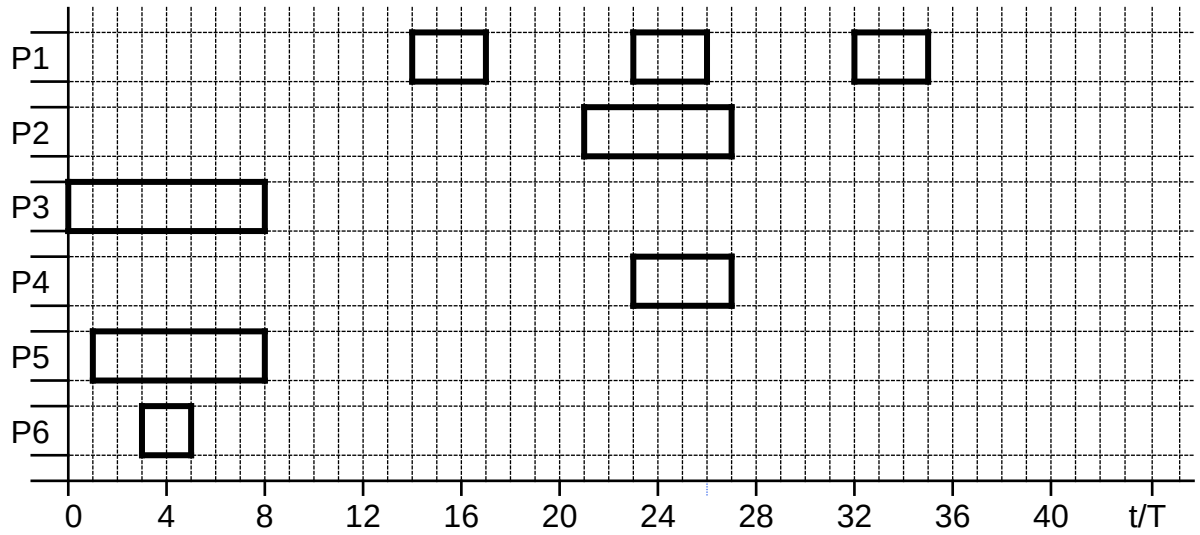


Diagramm BS-1.1: Sollzeitverlauf der Prozesse P1 bis P6

Prozess	P1	P2	P3	P4	P5	P6
Priorität	1 (hoch)	1	2	2	2	2 (niedrig)

Tabelle BS-1.2: Prioritätenverteilung

Hinweis: Bitte füllen Sie die nachfolgenden Aufgaben nach dem folgenden Schema aus:

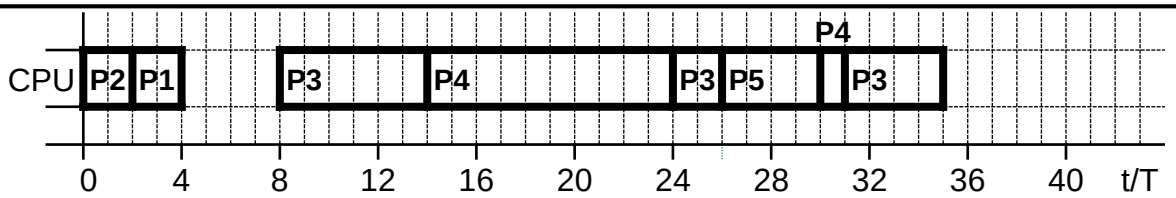


Diagramm BS-1.3: Lösungsbeispiel

Nachname, Vorname _____ Matrikelnummer _____

a) In der ersten Teilaufgabe sollen die Prozesse nicht-präemptiv nach ihren Prioritäten eingeplant werden. Bei Prozessen mit gleicher Priorität entscheidet der Zeitpunkt ihres Auftretens (Prio mit FIFO).

Tragen Sie den Verlauf der Prozessabarbeitung in das Diagramm BS-1.a ein.

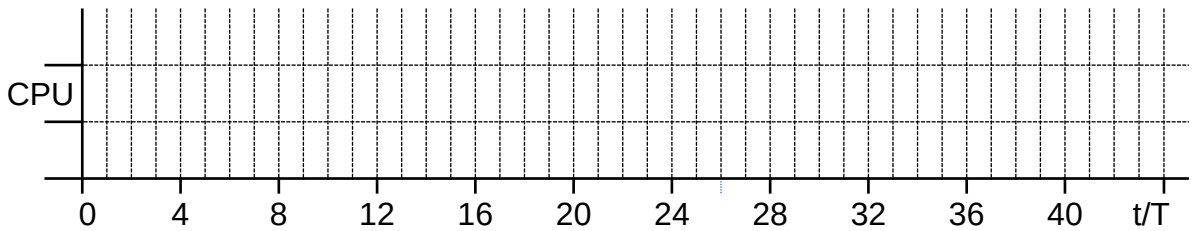


Diagramm BS-1.a: Prioritäten-Verfahren mit First In First Out (Prio mit FIFO)

b) In der zweiten Teilaufgabe soll die Einplanung der Prozesse ebenfalls präemptiv nach ihren Prioritäten erfolgen. Dabei muss jedoch für jedes Prioritätslevel ein Round-Robin-Verfahren (Prio mit RR) verwendet werden. Neue Prozesse werden nach dem Zeitpunkt ihres Auftretens (RR mit FIFO) auf die Zeitscheibe gelegt. Für die Zeitscheiben soll angenommen werden, dass diese unendlich viele Schlitze besitzen und so zu jedem Zeitpunkt ausreichend freie Schlitze vorhanden sind. Die Zeitschlitze besitzen eine Länge von 4T. Restzeiten können nicht übersprungen werden.

Tragen Sie den Verlauf der Prozessabarbeitung in das Diagramm BS-1.b ein.

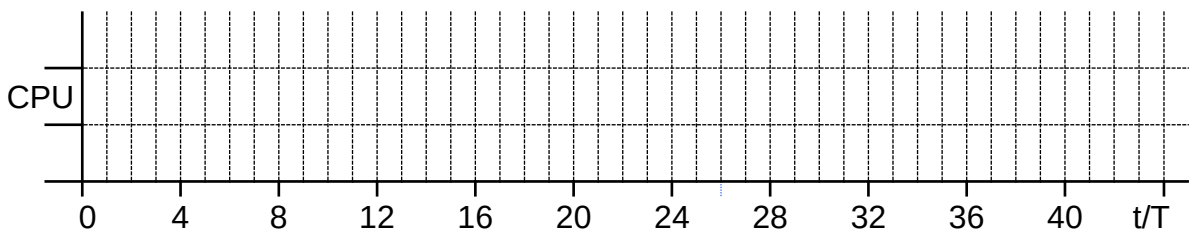


Diagramm BS-1.b: Prioritäten-Verfahren mit Round Robin (Prio mit RR)

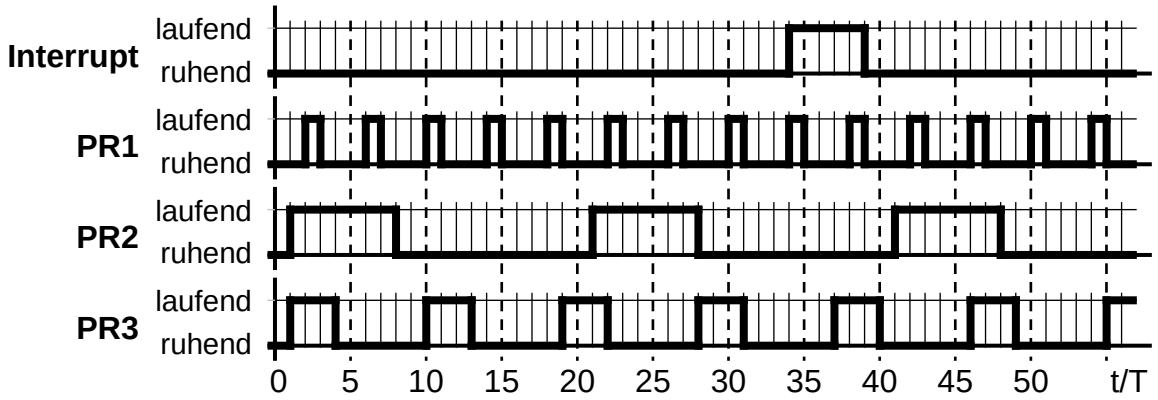
Nachname, Vorname _____ Matrikelnummer _____

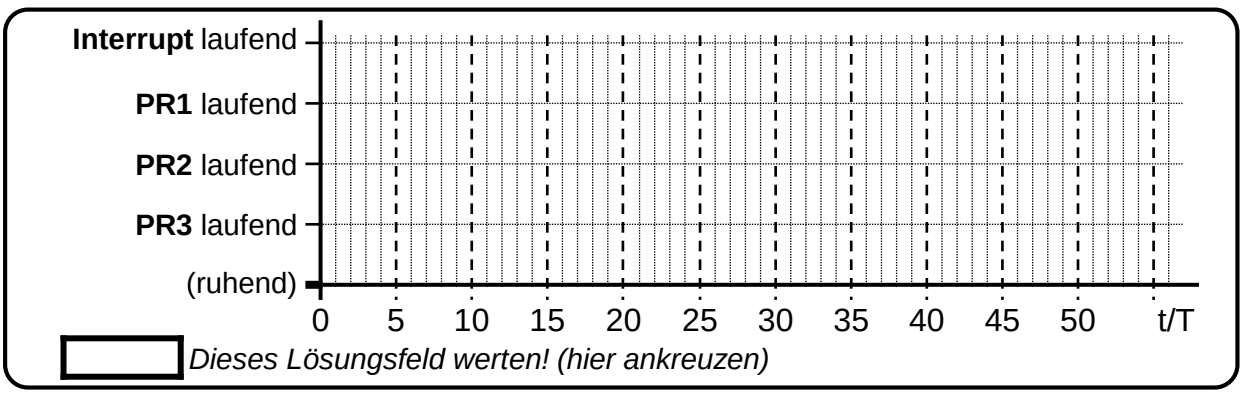
2. Asynchrone Programmierung

Drei periodische Prozesse (PR1 bis PR3) sollen mit dem Verfahren der asynchronen Programmierung nicht-präemptiv auf einem Einkernprozessor eingeplant werden. Der Prozess PR1 besitzt die höchste, der Prozess PR3 die niedrigste Priorität. Die Ausführung wird durch einen Interrupt unterbrochen. Die Einplanung der Interruptroutine erfolgt präemptiv. Nach dem Interrupt kehrt der Scheduler an die zuvor verlassene Stelle zurück.

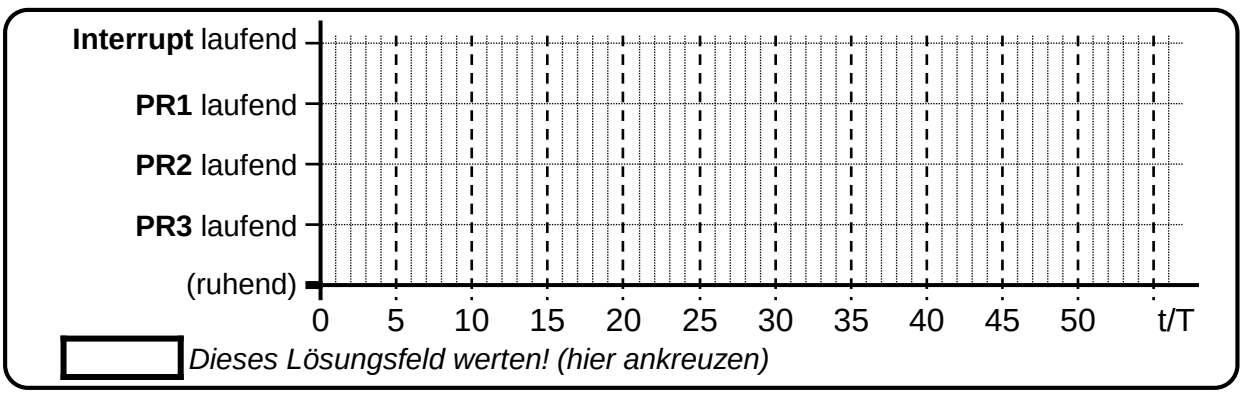
Tragen Sie in das unten angegebene leere Diagramm den Verlauf der Abarbeitung von Programmen und Interrupts ein.

Hinweis: Bei größeren Korrekturen verwenden Sie bitte das Ersatzfeld und markieren das zu wertende Lösungsfeld. Die Bewertung des Verlaufs erfolgt zeilenweise für jeden Prozess/Interrupt und unabhängig voneinander für die erste und zweite Hälfte des Diagramms





☐ *Dieses Lösungsfeld werten! (hier ankreuzen)*

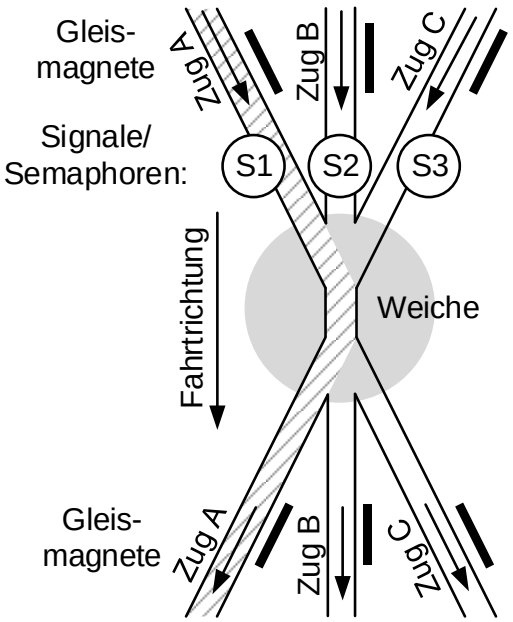


☐ *Dieses Lösungsfeld werten! (hier ankreuzen)*

Nachname, Vorname _____ Matrikelnummer _____

3. Semaphoren

Eine Weiche soll die Durchfahrt von drei Zugklassen (Zug A, Zug B, Zug C) durch eine Engstelle regeln. Da die Zugklasse A (Zug A) doppelt so häufig verkehrt wie die Zugklasse C (Zug C) und die Zugklasse B (Zug B) nur einmal verkehrt, soll die Weiche die Züge in folgender Reihenfolge passieren lassen: $AB\overline{ACA}$.



Entwickeln Sie mit Hilfe der drei Semaphoren S1, S2 und S3 für jede der drei Tasks (Züge) eine Anordnung von Semaphoroperationen. Geben Sie auch Initialwerte für die drei Semaphoren an.

Hinweis: Die Taskreihenfolge muss durch die Semaphoroperationen eindeutig festgelegt sein. Die für die Ausführung eines Tasks notwendigen Semaphoren sollen nur im Block verwendet werden. Beispielsweise würde ein Task X mit folgenden Semaphoroperationen

Task	X
Semaphoren-operationen ↓	P(S7)
	P(S7)
	P(S8)
	…
	V(S9)

nur starten, wenn alle drei benötigten Semaphoren (S7, S7, S8) gleichzeitig frei sind.

Semaphore:	**S1**	**S2**	**S3**
Initialwert:			

Task:	**A**	**B**	**C**
Semaphor-operationen ↓			

zu definierende Folge der Zugklassen: $AB\overline{ACA}$

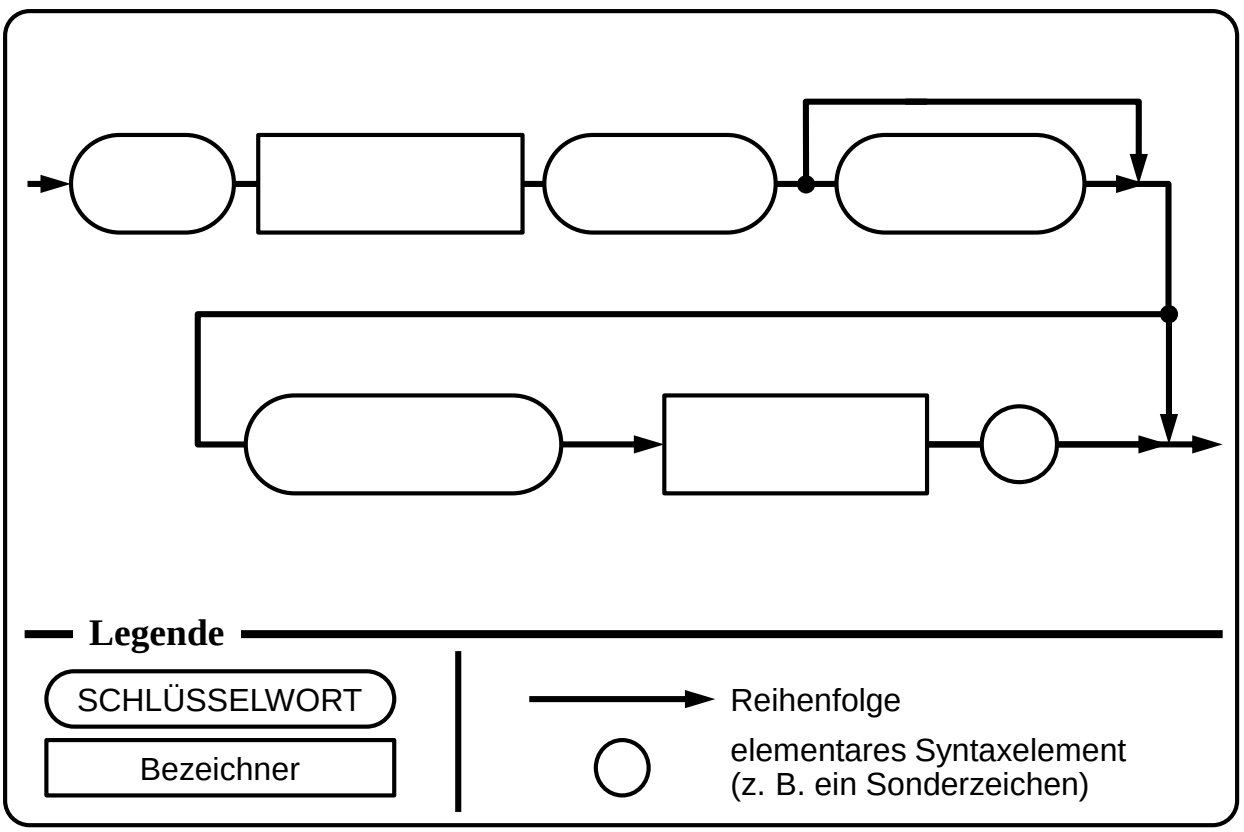
Nachname, Vorname _____ Matrikelnummer _____

4. PEARL

Eine Steuerung führt folgendes PEARL-Programm (PROBLEM) aus:

```
PROBLEM ;
   DCL Pufferzugriff SEMA GLOBAL PRESET 1;
   DCL Terminalzugriff SEMA;
   DCL Druckerzugriff SEMA GLOBAL;

   Puffer:    TASK ;
                  REQUEST Pufferzugriff ;
                  /* Folgt Arbeit mit Puffer */
                  RELEASE Terminalzugriff ;
              END;

   Terminal: TASK ;
                  REQUEST Terminalzugriff ;
                  /* Folgt Arbeit mit Terminal */
                  RELEASE Pufferzugriff ;
              END;
   …
```

Leiten Sie aus dem gegebenen Programmausschnitt die Schlüsselwörter, Bezeichner und Syntaxelemente des Syntaxgraphen für die Deklaration einer Semaphore in der Programmiersprache PEARL ab und tragen Sie diese in die untenstehenden sieben Platzhalter ein.

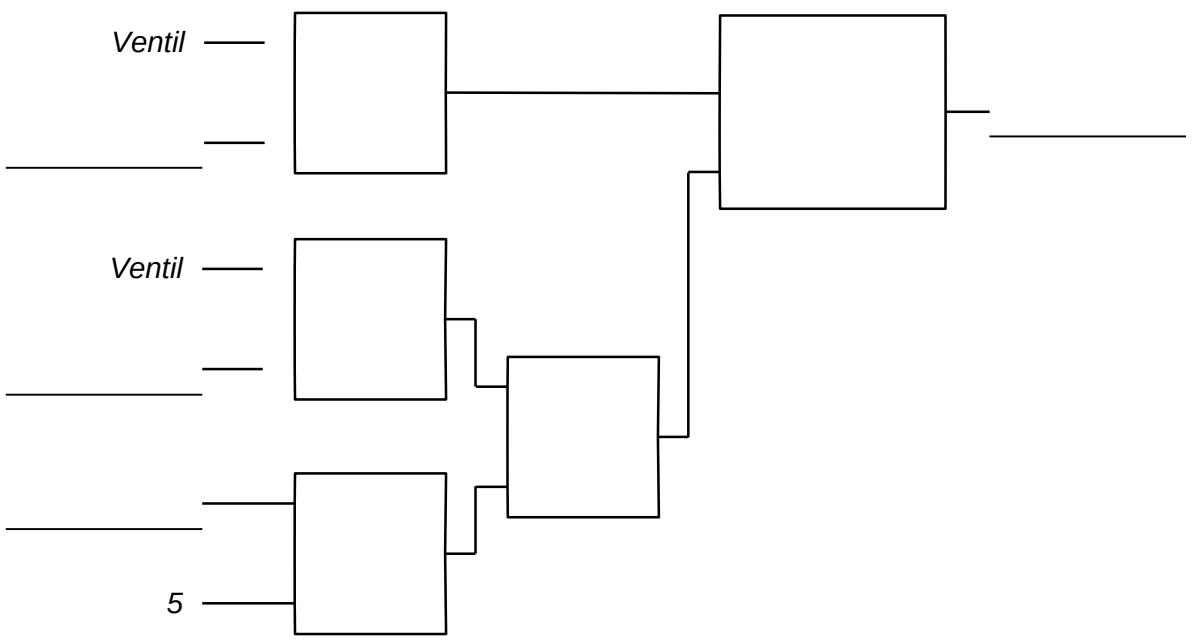


Nachname, Vorname _____ Matrikelnummer _____

5. IEC 61131-3: Funktionsbausteinsprache (FBS)

Mit einem *Taster* kann aus einem Spender Wasser abgefüllt werden. Mit dem *Taster* wird das *Ventil* des Spenders gesteuert. Wird der *Taster* betätigt (*Taster = 1*), während das *Ventil* geschlossen (*Ventil = 0*) ist, öffnet das *Ventil*. Wird der *Taster* betätigt, während das *Ventil* offen (*Ventil = 1*) ist, wird dieses geschlossen. Die Steuerung reagiert auf den *Taster* nur im Moment des Drückens. Des Weiteren soll der *Füllstand* des Wasserspenders (*Füllstand = 0..100*) nicht unter 5 [Prozent] fallen.

Vervollständigen Sie den untenstehenden Funktionsbausteinplan mit den fehlenden Variablennamen, Funktionsblockbezeichnungen und Verbindungen.



Nachname, Vorname _____ Matrikelnummer _____

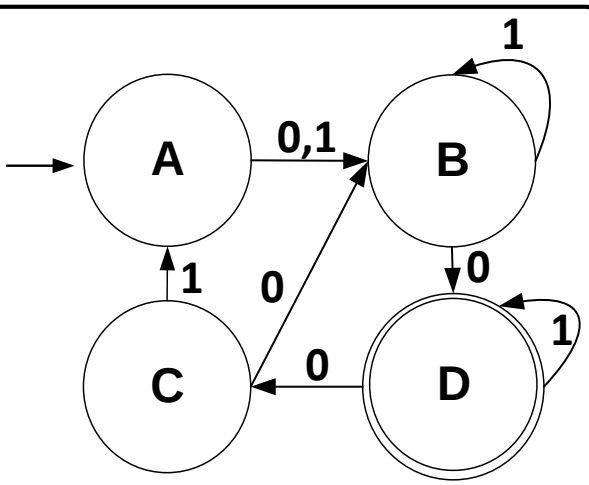
Aufgabe MSE Teil 1: Automaten

Aufgabe MSE1:
24 Punkte

1. Akzeptor

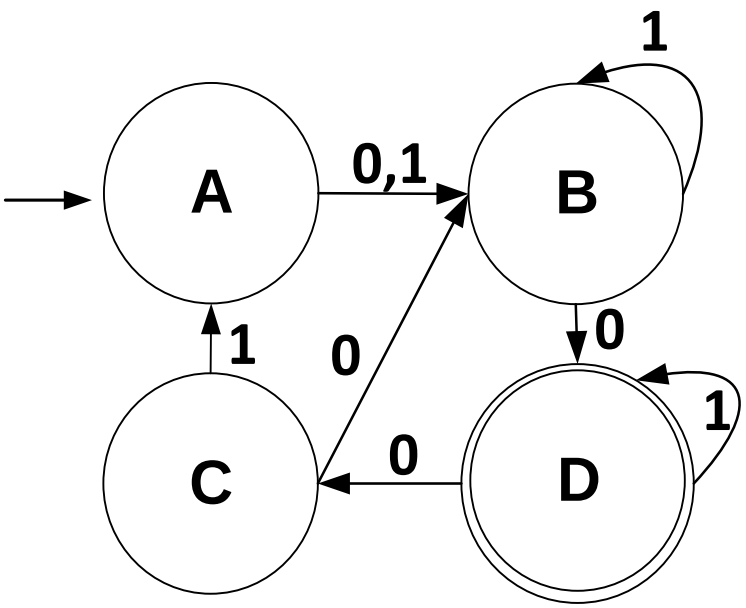
Gegeben sei folgender Automat. Geben Sie für die jeweilige Eingabe in den vorgesehenen Feldern an, ob der Automat die Eingabe akzeptiert und in welchem Zustand sich der Automat nach der Eingabe befindet.

Eingabe	Eingabe akzeptiert? (ja/nein)	Zustand nach Eingabe
0 0 0 0 0		
1 0 1 0 1		



2. Nicht-Deterministischer Automat

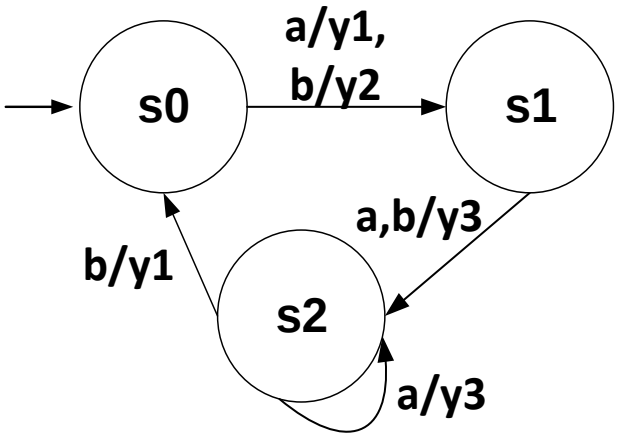
Ändern Sie den Automaten aus Aufgabe 1 in untenstehendem Feld so ab, dass sich sowohl für die Eingabe ‚0' als auch für die Eingabe ‚1' ein nicht-deterministischer Automat ergibt.



Nachname, Vorname _____ Matrikelnummer _____

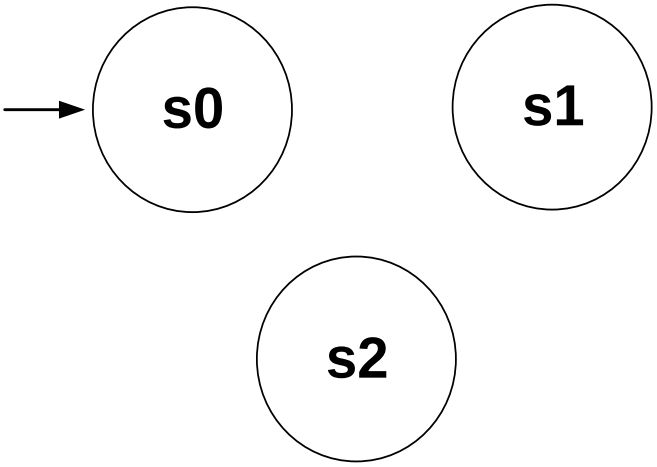
3. Moore-/Mealy-Automat

Gegeben ist folgender Automat. Bestimmen Sie, ob es sich um einen Moore- oder einen Mealy-Automaten handelt und wandeln Sie den Automaten in die äquivalente andere Form um (Moore → Mealy bzw. Mealy → Moore; Ergänzen Sie bei Bedarf zusätzliche Zustände).



Art des Automaten: _____

Umwandlung:



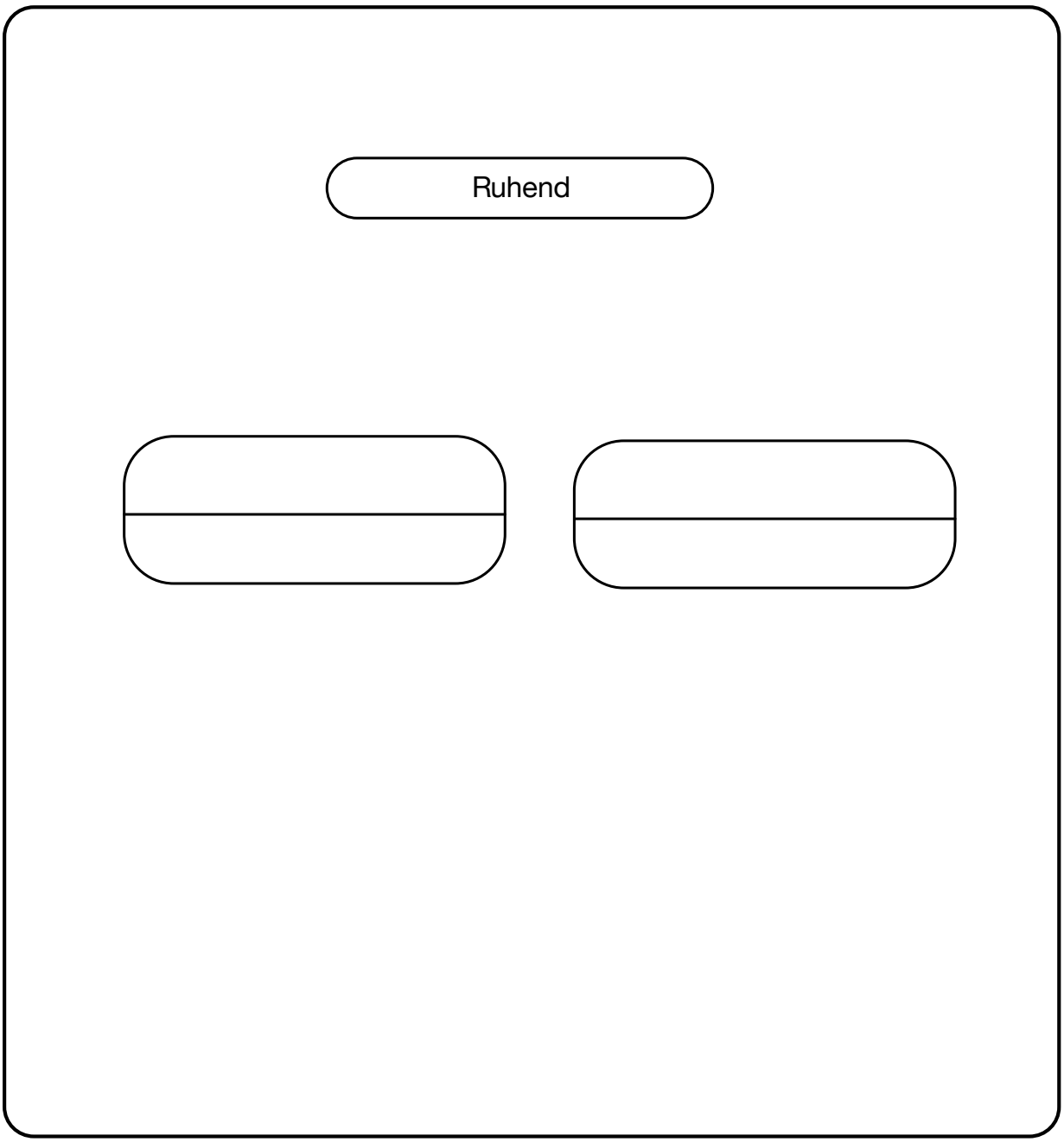
Nachname, Vorname

Matrikelnummer

4. Zustandsdiagramm

Der Funktionsablauf eines Förderbandes mit 2 Geschwindigkeitsstufen (Variable *Speed* mit den Werten 1 bzw. 2, Wert 2 entspricht schnell) soll als Zustandsdiagramm modelliert werden. Als erstes befindet sich das Band im Zustand *Ruhend*. Durch den Trigger *Fördern* geht das Förderband je nach Wert der Variable Speed in den Zustand *Langsam* bzw. *Schnell* über. Im jeweiligen Zustand wird die Aktion *MotorBewegen* durchgeführt. Durch den Trigger *Stopp* wird das Zustandsdiagramm verlassen.

Vervollständigen Sie das Zustandsdiagramm. Fügen Sie Start- und Endzustand ein. Zeichnen Sie keine zusätzlichen Zustände ein.



Nachname, Vorname _____ Matrikelnummer _____

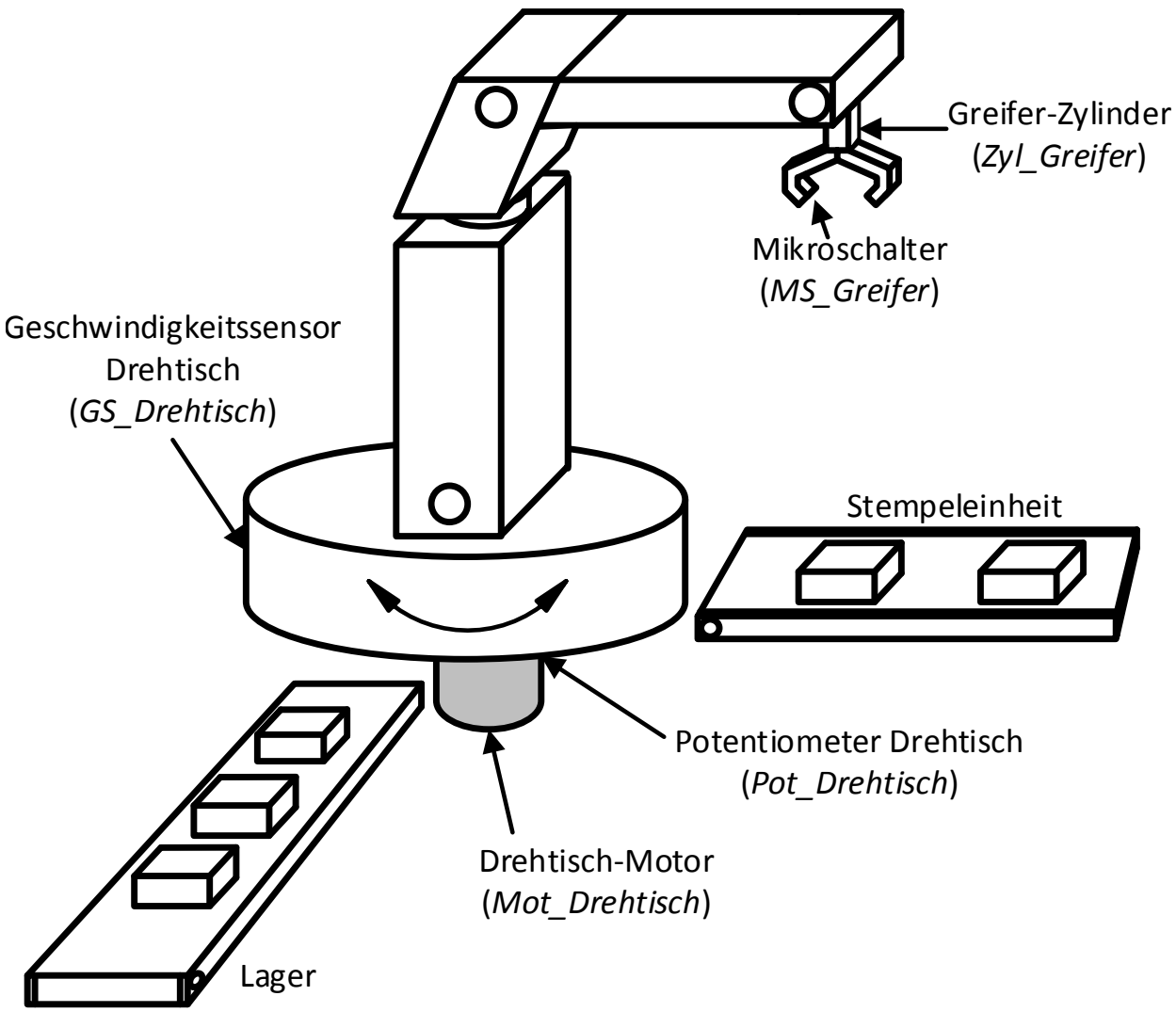
Aufgabe MSE Teil 2: Strukturierte Analyse/ Real-Time (SA/RT)

Aufgabe MSE2: 24 Punkte

Gegeben ist ein Fertigungssystem bestehend aus einem Lager, einem Kran und einer Stempeleinheit. Das unten abgebildete Kranmodul soll in den folgenden Aufgaben mittels Strukturierter Analyse/ Real-Time (SA/RT) modelliert werden. Das Kranmodul befördert Werkstücke vom Lager zur Stempeleinheit.

Dabei dient der Greifer dazu, das Werkstück (WS) zu greifen und am Stempel auszulassen. Der Greifer wird mittels eines Zylinders geöffnet/geschlossen. Ein Mikroschalter am Greifer dient zur Erkennung eines gegriffenen Werkstücks.

Zur Drehung des Krans dient der Drehtisch. Er wird von einem Motor bewegt. Zur Erkennung der Kranposition dient ein Potentiometer. Zur Erkennung der Krangeschwindigkeit ist zusätzlich ein Geschwindigkeitssensor angebracht.



Nachname, Vorname _____ Matrikelnummer _____

1. Datenflussdiagramm

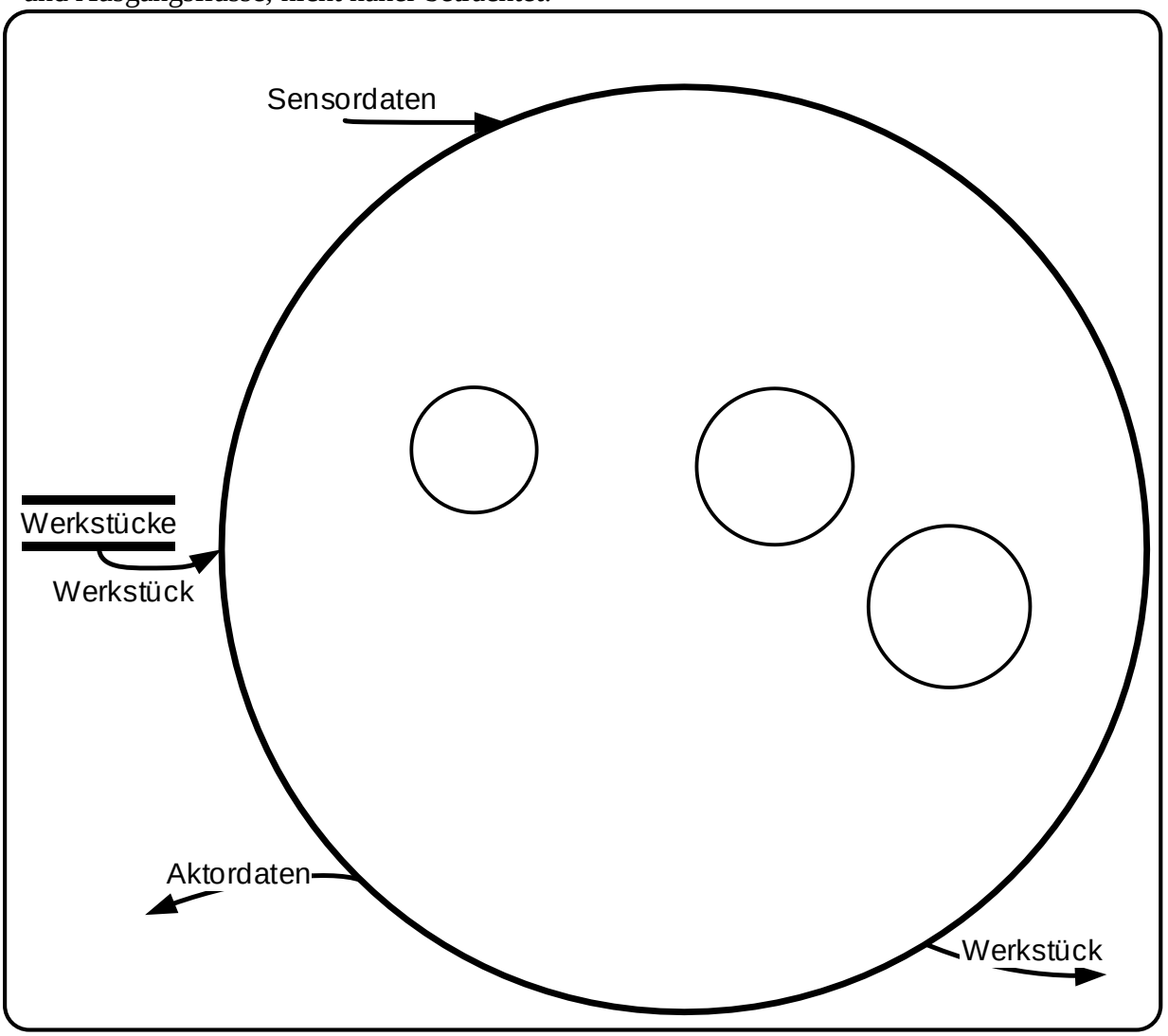
Es soll das Datenflussdiagramm des Prozesses *Werkstück mit Kran bewegen* modelliert werden. Der Prozess besteht aus den Subprozessen *Greife WS, versetze WS, lasse WS aus*. Folgende Flüsse sollen dabei betrachtet werden:

Materialfluss: Das am Lager positionierte WS (*WS positioniert am Lager*) wird gegriffen. Das *gegriffene WS* wird dann versetzt. Das *versetzte WS* wird dann vom Greifer ausgelassen. Damit ist das WS am Stempel positioniert (*WS positioniert bei Stempel*).

Sensordaten: Von dem übergeordneten Prozess fließen die *Sensordaten des Greifers* (notwendig zum Greifen und zum Auslassen des WS) und des *Drehtisches* (notwendig zum Versetzen des WS) zu den entsprechenden Subprozessen.

Aktordaten: Die Aktordaten setzen sich aus den *Aktordaten des Greifers* und den *Aktordaten des Drehtisches* zusammen.

Vervollständigen Sie das untenstehende Datenflussdiagramm des beschriebenen Prozesses „*Werkstück mit Kran bewegen*“. **Hinweis**: Das übergeordnete DFD 0 wird, bis auf die Ein- und Ausgangsflüsse, nicht näher betrachtet.



Nachname, Vorname _____ Matrikelnummer _____

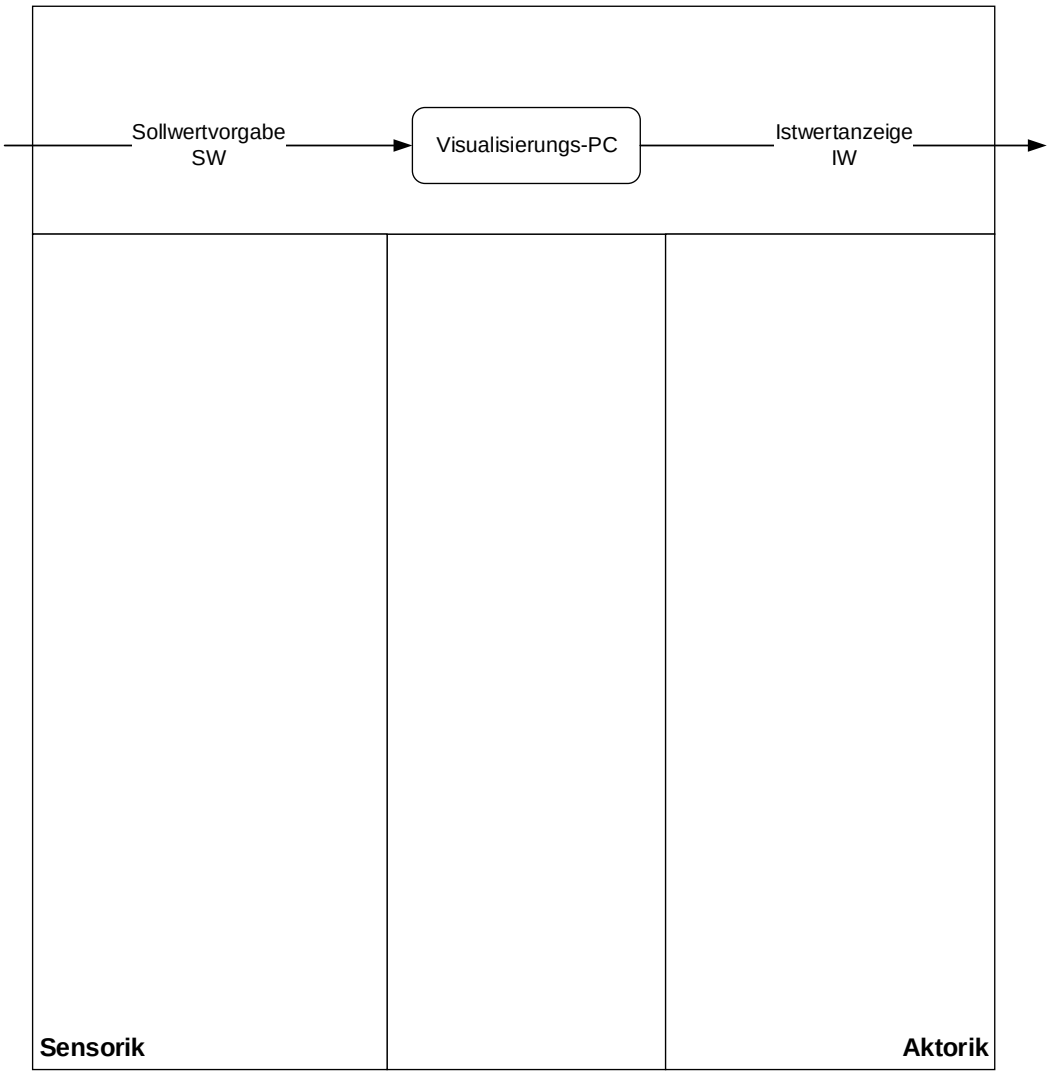
2. Architekturflussdiagramm

Zur Steuerung des Krans werden 3 Mikrocontroller eingesetzt. Ein Mikrocontroller (µC1) ist für die Sensorik und Aktorik des Greifers verantwortlich, ein zweiter Mikrocontroller (µC2) entsprechend für den Drehtisch. Der dritte Mikrocontroller (µC3) dient dazu, die Soll- /Ist-werte (SW/IW) mit dem Visualisierungs-PC auszutauschen. Dazu werden die SW/IW mit den beiden anderen Mikrocontrollern ausgetauscht. Um die Drehbewegung des Krans einzuleiten, wird zudem der Istwert von µC1 an µC2 übergeben. Folgende Sensorik/Aktorik soll dabei betrachtet werden:

- **Sensorik:** *Mikroschalter, Potentiometer, Geschwindigkeitssensor*
- **Aktorik:** *Greifer-Zylinder, Drehtisch-Motor*

Modellieren Sie das Architekturflussdiagramm mit den Architekturmodulen sowie die entsprechenden Flüsse (Flüsse von der Umgebung zu den Sensoren bzw. von den Aktoren zu der Umgebung müssen nicht modelliert werden).

Hinweis: Verwenden Sie die Bezeichnungen aus der Abbildung auf Seite 17.



Nachname, Vorname _____ Matrikelnummer _____

Aufgabe C: Programmieren in C

Aufgabe C:
96 Punkte

1. Datentypen, Ein-/Ausgabe und boolesche Algebra

a) Erstellen Sie die Definition und Initialisierung der folgenden Variablen mit sinnvollen Datentypen und aussagekräftigen Anfangsbuchstaben der Variablenbezeichnung, um Daten über ein Fahrzeug speichern zu können.

- Fahrzeug-Kennzeichen (z. B. „AA-BB 0000“, max. 10 Stellen)
- Buchstabe der benötigten Führerscheinklasse (z. B. „A“ – Motorrad, „B“ – PKW, „C“ – LKW) für das Fahrzeug
- Geschwindigkeit des Fahrzeugs auf eine Nachkommastelle genau

Geben Sie zudem die Variable der Geschwindigkeit mittels formatierter Ausgabe mit genau einer Nachkommastelle aus.

```
_________  _________________ = ___________________

_________  _________________ = ___________________

_________  _________________ = ___________________

// Formatierte Ausgabe der ermittelten Geschwindigkeit

_________(__Ihre Geschwindigkeit: _____________________)__
```

b) Bestimmen Sie das Ergebnis folgender Ausdrücke im Dezimalsystem, welches sich aus mathematischen, logischen bzw. bitweisen Operatoren berechnet.

Int-Variablen: X = 9; Y = 3; Float-Variable: Z = 0.75;
Char-Variable: A = 0x12; Int-Zeiger: pi=&X;

X & Y | (int) A	
(X / Z) != (X + Y)	
(*pi & (char) Y)	

Nachname, Vorname

Matrikelnummer

2. Kontrollstrukturen

a) Ein Programm zur Steuerung einer Geschwindigkeitsüberwachung soll um die Möglichkeit zur Auswertung der ermittelten Geschwindigkeitsverstöße erweitert werden. Dabei soll neben Anzahl an Verstößen auch der Durchschnitt der überschrittenen Geschwindigkeiten ermittelt werden.

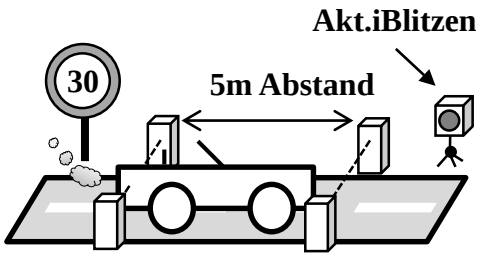
- Mit Hilfe einer geeigneten Schleife sollen alle Geschwindigkeits-Messwerte aus dem float-Array `fMesswert` mit der Größe `MAX` nacheinander gelesen werden. Der letzte Messwert ist erreicht wenn das Feld des Arrays den Wert -1 enthält oder das letzte Feld des Arrays gelesen wurde.
- Falls die Höchstgeschwindigkeit `iHoechstGesch` nicht überschritten wurde, kann sofort mit dem nächsten Messwert fortgefahren werden, andernfalls muss die Variable `iAnzGeschwUeber` um eins erhöht und der Messwert zu der Variable `fSumGeschwUeber` addiert werden.
- Berechnen Sie nach der Auswertung des letzten Messwerts der Datenbank die durchschnittliche Geschwindigkeit der Fahrzeuge, welche die Höchstgeschwindigkeit überschritten haben und lassen Sie diesen Wert von der Funktion zurückgeben.

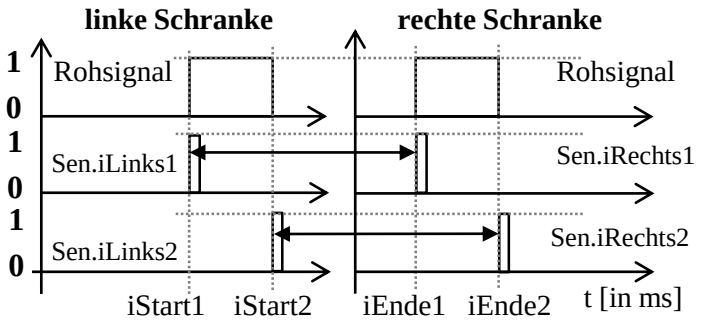
```
#include <stdio.h>
#define MAX 6
float durschGeschwBeiUebersch()
{
    float fMesswert[MAX]  = {53.0, 65.3, 47.1, -1, -1, -1};
    int   i               = 0;   // Laufvariable
    int   iHoechstGesch   = 50;
    int   iAnzGeschwUeber = 0;
    float fSumGeschwUeber = 0;
```

```
}
```

Nachname, Vorname _____ Matrikelnummer _____

3. Zyklische Programmierung einer Blitzeranlage





Die Firma ISA GmbH stellt eine Blitzanlage her, welche aus zwei Lichtschranken besteht, die in einem Abstand von 5 Metern aufgebaut werden. Die Blitzersteuerung erkennt drei Fälle: Die Geschwindigkeit war zu hoch, in Ordnung oder die Berechnung ist fehlgeschlagen. Zur Fehlererkennung wird die Geschwindigkeit zweimal berechnet: Einmal anhand der beiden steigenden und einmal mit Hilfe der fallenden Signalflanken (vgl. oben rechts). Man geht davon aus, dass das Fahrzeug mit konstanter Geschwindigkeit durch die Strecke fährt. Bei einer Geschwindigkeit über 30 km/h soll der Blitzer ausgelöst werden.

Implementierung des Zustands (1)„Messung starten": Der Übergang nach (1) soll durch den Sensor Sen.iLinks1, welcher links eine steigende Flanke erkennt, ausgelöst werden. Nun soll die zugehörige Systemzeit in die Variable iStart1 gespeichert werden. Beim Herausfahren des Autos aus der linken Schranke wird eine fallende Flanke (Sen.iLinks2) erkannt und die Zeit soll in die Variable iStart2 gespeichert werden. Der Übergang zu „Messung stoppen" (2) wird ausgelöst durch eine steigende Flanke der rechten Lichtschranke (Sen.iRechts1).

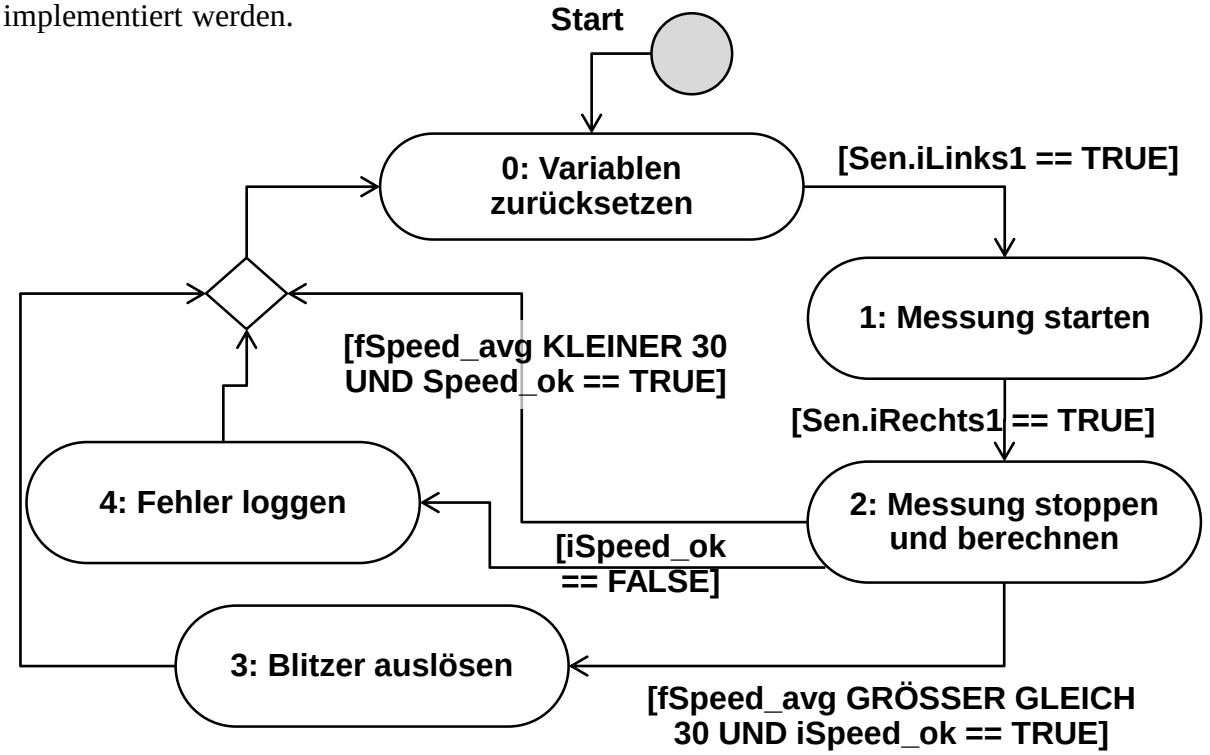
Implementierung des Zustands (2)„Messung stoppen und berechnen": Zunächst werden zwei Werte iEnde1/iEnde2 analog zum Zustand „Messung starten" gespeichert. Durch die zwei Wertepaare können nun zwei Geschwindigkeiten (fSpeed1/fSpeed2) berechnet werden. Anschließend wird in fSpeed_avg der Durchschnitt abgelegt. (*Fortsetzung nächste Seite*)

Aktoren:	
Akt.iBlitzen	Löst nach Setzen von TRUE den Blitzer aus
Sensoren/Merkvariablen:	
Sen.iLinks1 / Sen.iLinks2	Erkennt steigende (1) / fallende (2) Flanke links und gibt 1 aus
Sen.iRechts1 / Sen.iRechts2	Erkennt steigende (1) / fallende (2) Flanke rechts und gibt 1 aus
fSpeed1 / fSpeed2	Speichert Geschwindigkeiten 1 und 2
iStart1/ iStart2	Speichert Startzeiten 1 und 2
iEnde1/ iEnde2	Speichert Endzeiten 1 und 2
fSpeed_avg	Speichert Berechnung der Durchschnittsgeschwindigkeit
iSpeed_ok	Boole-Wert mit 1 für korrekte und 0 für fehlerhafte Messung
plcZeit	Systemlaufzeit der Steuerung (in ms)
state	Speichert den aktuellen Zustand

Nachname, Vorname | Matrikelnummer

Dann soll mit einer IF-Bedingung geprüft werden, ob der Absolutwert der Differenz beider Messungen kleiner bzw. gleich (iSpeed_ok=1) oder größer als 3 km/h war (iSpeed_ok=0). Nutzen Sie zur Berechnung des Absolutwerts die abs-Funktion (z.B.: `x=abs(y);`). Ist die Geschwindigkeit korrekt, wird bei Überschreitung (fSpeed_avg > 30 km/h) in (3) geblitzt und bei Unterschreitung direkt zurück in den Initialzustand (0) gewechselt.

Vorgehen: Übertragen Sie das Zustandsdiagramm und die oben beschriebene Funktionalität in den C-Code auf den folgenden Seiten. Benutzen Sie die Variablen auf der vorherigen Seite. Beachten Sie, dass der Code zyklisch ausgeführt wird. Der Kopf des Programms mit Deklaration und Initialisierung aller Variablen ist bereits unten gegeben. **Hinweis:** Der Zustand „Fehler loggen“ (4) muss nicht implementiert werden, aber die zugehörige Variable iSpeed_ok korrekt gesetzt und die richtige Transition nach (4) implementiert werden.




```
#include "eavar_blitzer.h"
#include <math.h>          // Fuer Verwendung von abs()

struct SData Sen;          // Sensorvariablen
struct AData Akt;          // Aktorvariablen
unsigned int state=0;      // Zustandsvariable (Start in 0)
unsigned int plcZeit=0;    // in ms (1000 entspricht 1 Sek.)
float fSpeed1=0,fSpeed2=0;// Speicher für Rechenergebnisse der
                           // beiden Geschwindigkeiten
float fSpeed_avg=0;        // Durchschnitt
int iStart1=0, iStart2=0; // Speicher für Startzeiten
int iEnde1=0, iEnde2=0;    // Speicher für Endzeiten
int iSpeed_ok=0;           // Flag ob die Messung korrekt (1)
                           // oder fehlerhaft (0) war
```

Nachname, Vorname _____ Matrikelnummer _____

```
switch(state)
{
  _____________ // State 0: Init
  reset_all();  // Alle Variablen auf 0 setzen

  _____________________________________;  //Uebergang 0->1

  _____________

  _____________ // State 1: Messung_starten

  ___________________________________________ // Messung 1

  ____________________________________________________________
  // Messung 2

  ____________________________________  //Uebergang 1->2

  _____________

  ___________ //State 2: Messung stoppen & berechnen

  ___________________________________ // Messung 1 beenden

   if(Sen_iRechts2==1 && !fSpeed_avg)
         {   // Messung 2 beenden und berechnen

        ____________________________________________________

        ____________________________________________________

        ____________________________________________________

         ____________________________________________________

        _____________________________________//Messung korrekt?

        _____________________________________//iSpeed_ok setzen

         }
// Fortsetzung naechste Seite!
```

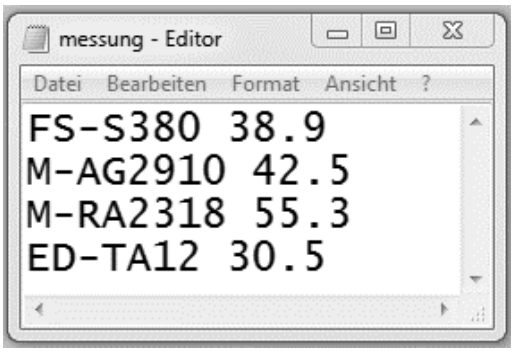
Nachname, Vorname _____ Matrikelnummer _____

```
// Zustandsuebergaenge 2->3, 2->0 und 2->4

        _____________________________________________

        _____________________________________________

        _____________________________________

        _________________

___________// State 3: Blitzen

        ______________________

        ______________________ // Uebergang 3->0

        _________________

(...) //State 4: Fehler setzen (nicht dargestellt)
```

Nachname, Vorname _____ Matrikelnummer _____

4. Einlesen der Verkehrssünder in eine Datenbank

Die zuvor mit Hilfe der Blitzersteuerung aufgenommen Fahrzeuge sollen nun in eine Datenbank übernommen werden. Um eine hohe Kompatibilität zu gewährleisten, speichert die Steuerung alle Verkehrssünder in die Textdatei „messung.txt“. Um die Daten geordnet in ein C-Programm einzulesen, soll zunächst ein neuer Datentyp angelegt werden.



Anschließend sollen die Daten in ein Array dieses Datentyps aus der Datei „messung.txt“ übernommen werden.

a) Legen Sie einen neuen Datentyp mit dem Namen „MESSUNG“ an und definieren Sie folgende Variablen in der Struktur:

- Kennzeichen (max. 10 Zeichen)
- Geschwindigkeit (mind. 1 Nachkommastelle)

```
#include <stdio.h>
#include <string.h>
```

Nachname, Vorname _____ Matrikelnummer _____

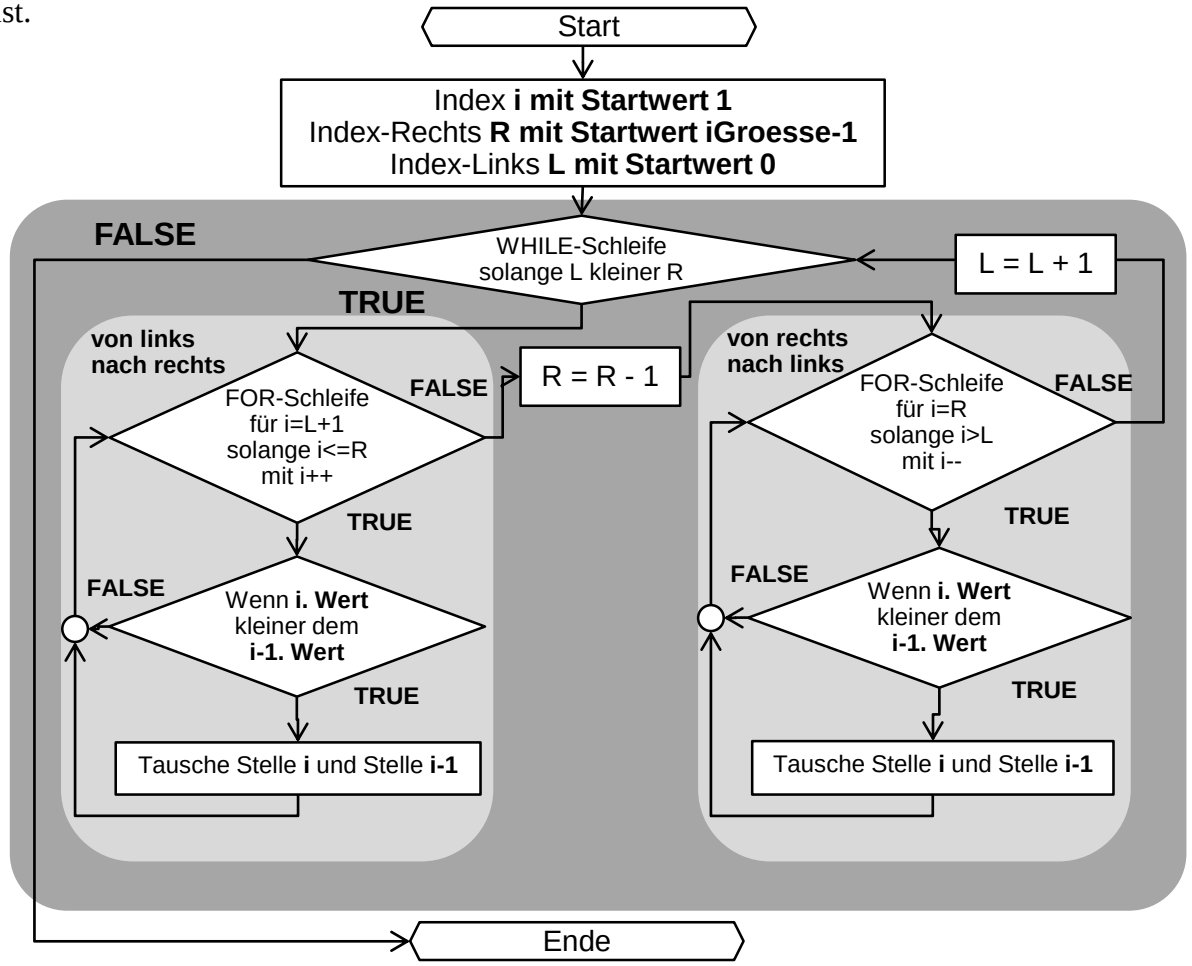
b) Vervollständigen Sie nun die main-Funktion zum Einlesen der Messung. Legen Sie dabei ein Array vom Typ MESSUNG an, welches 20 Einträge speichern kann. Öffnen Sie anschließend die Datei „messung.txt“ mittels eines FILE-Handle mit der Bezeichnung „pEingabe“ so, dass sie vom Programm nur gelesen werden kann. Um alle Daten einzulesen, ist eine Schleife notwendig, die bis zum Ende der Datei läuft. Verwenden Sie in der Schleife zwei geeignete FSCANF-Befehle um erst das Kennzeichen und dann die Geschwindigkeit auszulesen. Schließen Sie schlussendlich den Dateizugriff nach dem Ende der Schleife wieder.

```
  int main()
{   //Variablendeklaration
    int i=0, j=0;


    _____________________________________________
    //Öffnen der Datei messung.txt


    ____________________________________________________________
    //Schleife bis das Ende der Datei erreicht ist


    ____________________________________________________
    {
      if(_____________________________________________________==0)
        {
          printf("\nFehler beim Einlesen"); return 1;
        }
      if(_____________________________________________________==0)
        {
          printf("\nFehler beim Einlesen"); return 1;
        }
        i++;
    }
    //Schließen der Datei

    ________________________________________
    return 0;}
```

Nachname, Vorname _____ Matrikelnummer _____

5. Sortieren mittels Shaker-Sort

Zur Ermittlung der schlimmsten Tempo-Sünder soll eine Funktion implementiert werden, welche das Array vom Typ MESSUNG mit einer beliebigen Arraygröße „iGroesse“ sortieren kann. Hierbei sollen die Messungen mit der Geschwindigkeit *aufsteigend* (also höchste Geschwindigkeit zuletzt) sortiert werden. Hierfür soll der sog. *Shaker-Sort-Algorithmus* implementiert werden, welcher im Folgenden ausführlich in einem Flussdiagram beschrieben ist.



Zum weiteren Verständnis der Wirkungsweise des *Shaker-Sort-Algorithmus* ist im Folgenden ein Beispiel für die Zahlenreihe 1-4-3-2 (iGroesse=4 ➔ R=3, L=0) durchgeführt. „RW“ steht für einen Richtungswechsel. Die Plätze im Kästchen werden im aktuellen Schritt verglichen:

[1 4] 3 2 (i=1), 4 kleiner 1 ? ➔ Nein → i=i+1; i <= R ? Ja

1 [4 3] 2 (i=2), 3 kleiner 4 ? ➔ Tausch! ➔ 1 3 4 2 → i=i+1; i <= R ? Ja

1 3 [4 2] (i=3), 2 kleiner 4 ? ➔ Tausch! ➔ 1 3 2 4 → i=i+1; i <= R ? Nein, R=R-1=2, **RW**

1 [3 2] **4** (i=2), 2 kleiner 3 ? ➔ Tausch! ➔ 1 2 3 4 → i=i-1; i > L ? Ja

[**1 2**] 3 **4** (i=1), 2 kleiner 1 ? ➔ Nein → i=i-1; i > L ? Nein, L=L+1=1, **L<R? Ja**

1 [**2 3**] **4** (i=2), 3 kleiner 2 ? ➔ Nein → i=i+1; i <= R ? Nein, R=R-1=1 , **RW**

→ i=R=1, i > L? Nein L=L+1=2 → **L<R? Nein** ➔ Abbruchkriterium erfüllt (R=1<L=2)

Nachname, Vorname _____ Matrikelnummer _____

Vervollständigen Sie nun unten die Funktion „shakerSort“ entsprechend der zuvor im Flussdiagramm beschriebenen Funktionsweise. Die Funktion wird mittels *call-by-reference* aufgerufen und bekommt einen Zeiger auf das Array übergeben. Sie sortiert also direkt im Array vom Typ MESSUNG und gibt keinen Rückgabewert zurück.

Die verwendete Schleife benötigt die Größe des Arrays, welche durch die int-Variable iGroesse an die Funktion per Kopie (*call-by-value*) übergeben wird.

Die Arrayelemente sollen nach der Struct-Variable für die Geschwindigkeit *aufsteigend* sortiert werden (z. B. *fSpeed*).

Es sei des Weiteren bereits eine Funktion „*tausche*“ implementiert, die nicht dargestellt ist. Die Funktion kann zwei Arrayelemente vom Typ MESSUNG vertauschen. Verwenden Sie diese durch einen korrekten Funktionsaufruf mittels *call-by-reference*, indem Sie ihr die beiden Tauschpartner (also die entsprechenden Arraystellen) als Adresse übergeben.

Hinweis: Beachten Sie die zugehörigen Kommentare im Code zur Vervollständigung.

```
________ shakerSort(_______________________________)
{
  int R = iGroesse-1, L = 0, i = 1;

  ___________________ // Schleife
  {

      for (_____________________) // nach rechts

        ________________________________________

           ________________________________________

      _______________

      for (_____________________________) //nach links

        ________________________________________

           ________________________________________

      _______________
    }
}
```

Nachname, Vorname _____ Matrikelnummer _____

6. Funktions-Zeiger

Im folgenden Programm sind nun zwei unterschiedliche Sortierfunktionen enthalten. Beide sollen ein int-Array beliebiger Größe sortieren. Eine Funktion sortiert das Array aufsteigend (sort_up) und die andere absteigend (sort_down). Das Array wird am Ende des Programms ausgegeben, die zugehörige Funktion „Ausgabe“ wird jedoch nicht betrachtet.

Vervollständigen Sie zunächst die Funktionsköpfe der beiden Funktionen „sort_up“ und „sort_down“, so dass die richtigen Datentypen übergeben werden können.

Um für die Auswahl der richtigen Funktion keine Kontrollstruktur implementieren zu müssen, soll ein Funktionszeiger-Array erstellt werden, welches die beiden Funktionen „sort-up“ und „sort-down“ enthält. Nach dem Einlesen der Auswahl (*iWahl)* soll dann durch einen geeigneten Aufruf des Funktionszeigers die entsprechende Sortierfunktion aus dem Zeiger-Array ausgewählt werden und ihr die nötigen Daten übergeben werden.

```
#include <stdio.h>

________ sort_up(______ iArray,______ iGroesse)
{
    ...
}

________ sort_down(______ iArray,_______ iGroesse)
{
    ...
}

int main()
{
    //Variablendeklaration
    int iArray[5]={10,8,14,4,16};
    int i=0, iWahl=0, iLaenge=5;

    ____________________________________________________________

    //Einlesen Benutzerwahl
    printf("Sortierrichtung: [0]Aufsteigend [1]Absteigend");
    scanf("%i",&iWahl);

    //Funktionsaufruf

    ____________________________________________________________

    printf("\nErgebnis:");
    Ausgabe(&iArray[0],5);
    return 0;
}
```