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

Vorname:	
Nachname:	
Matrikelnummer:	

Aufgabe	Erreichte Punkte
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
∑	

Prüfung –Informationstechnik

Sommersemester 2020

31.08.2020

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält **26** nummerierte Seiten inkl. Deckblatt.

Bitte prüfen Sie die Vollständigkeit Ihres Exemplars!

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

Punkte

1. Umrechnung zwischen Zahlensystemen

Wie lautet das Ergebnis folgender Gleichung im Hexadezimalsystem?
$(0F)_{16}$ xor $(31)_{16}$

2. IEEE 754 Gleitkommazahlen

Rechnen Sie die Dezimalzahl $(-2{,}625)_{10}$ in eine Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) mit folgender Formatierung um:

V e (3 Bit) M (4 Bit)

Vorzeichen: **V = 1**

Mantisse (Binärzahl und normalisiert)

M_2=

Exponent

E =

Bias und biased Exponent

B=

e =

Vollständige Gleitkommazahl nach gegebener Formatierung

Multiplizieren Sie folgende Gleitkommazahl mit $(2)_{10}$. Wie lautet das Ergebnis?

1	0	1	0	1	0	0	1

V e (3 Bit) M (4 Bit)

3. Querparität

Folgende Nachricht wurde mittels ungerader Parität gegen Übertragungsfehler geschützt. Rekonstruieren Sie die fehlenden Daten und beantworten Sie nachfolgende Frage.

0		0	**0**
1		1	**0**
0		1	**1**

Wie viele Bits könnten mit Querparität fehlerhaft übertragen worden sein, ohne erkannt zu werden? (z.B. 1 oder 4 Stk.)

Punkte

4. Logische Schaltungen und Schaltbilder

a) Gegeben sei nebenstehende Wahrheitstabelle (Tabelle 4.1). Erstellen Sie das zugehörige Schaltbild der DNF.

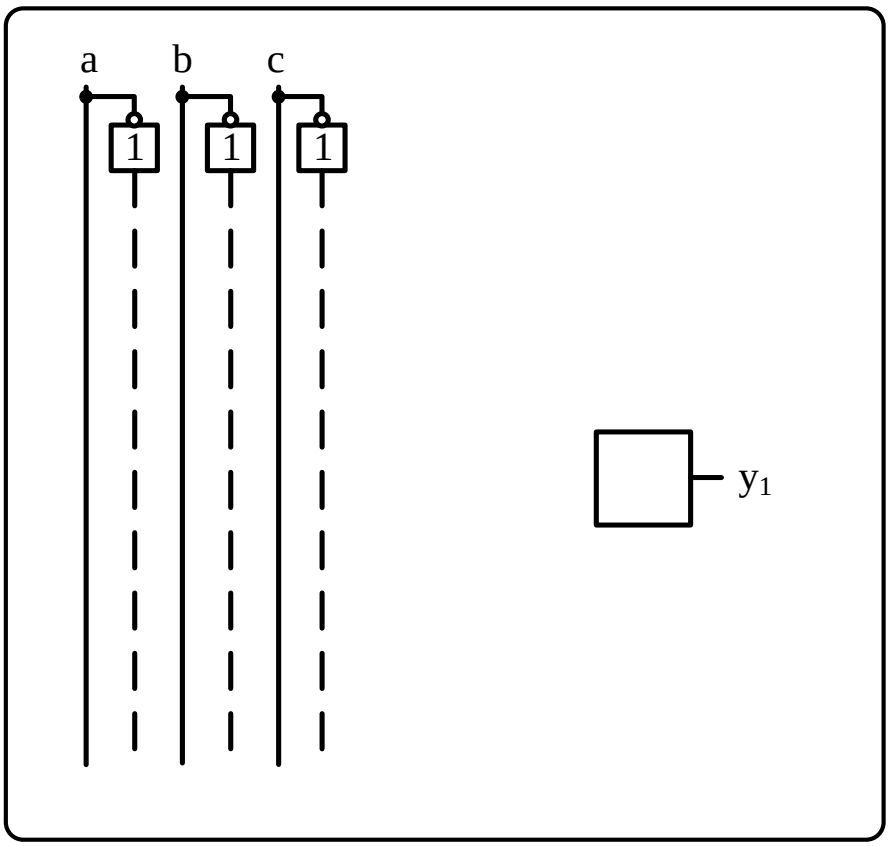


Tabelle 4.1:
Wahrheitstabelle

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

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

( ) Zweikanal-Multiplexer mit Selektionseingang „a“

( ) NAND-Gatter für 3 Eingänge

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

( ) Negiertes XOR-Gatter für 3 Eingänge

Punkte

5. Flip-Flops

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

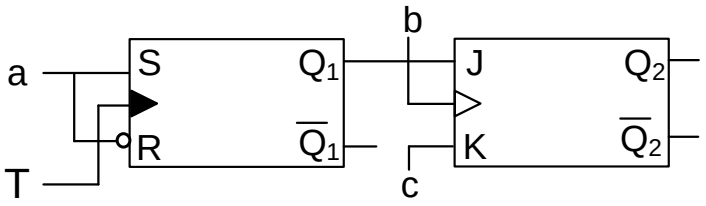
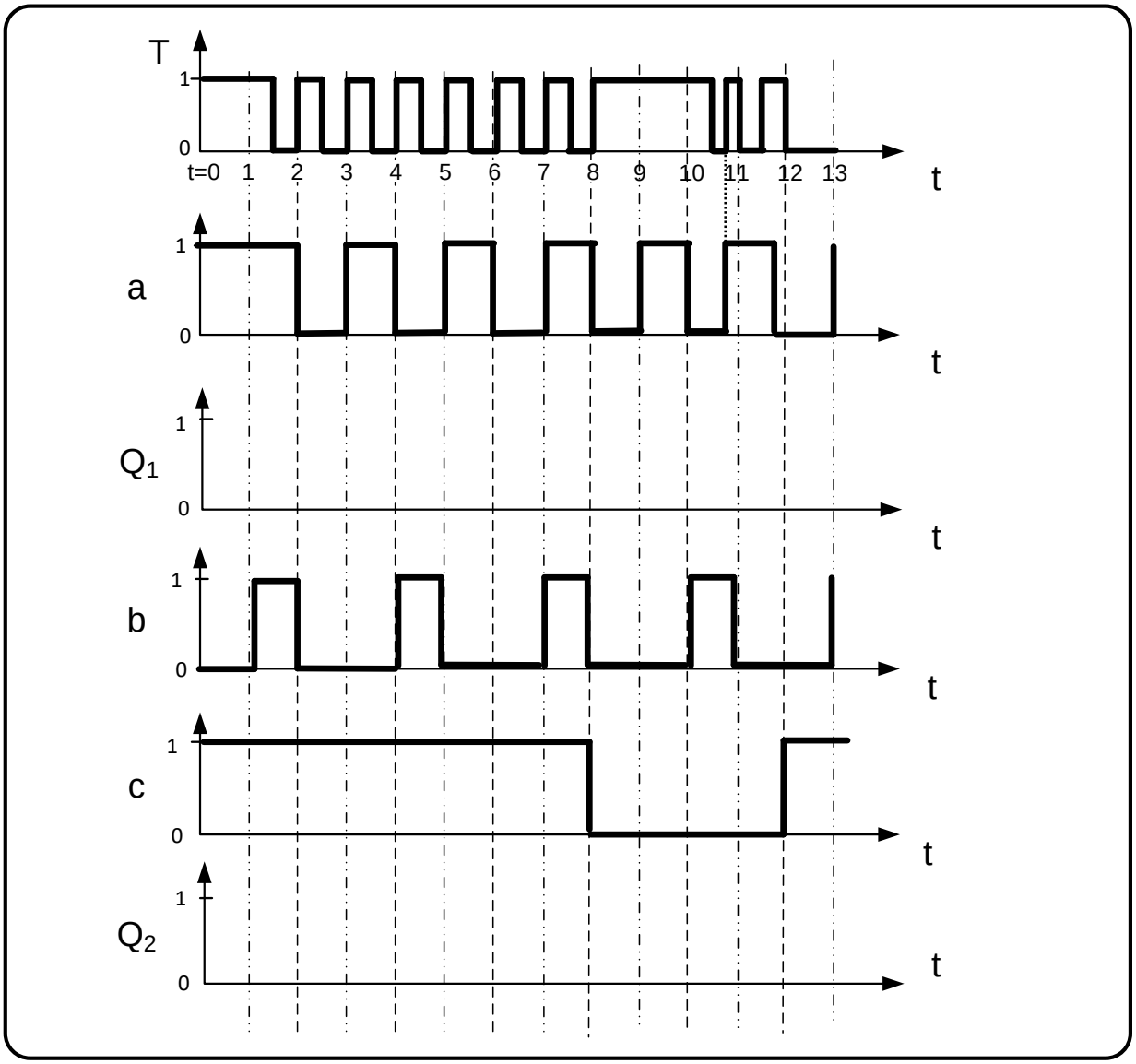


Bild 5.1: *MS-FF*

Bei 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.



Punkte

6. MMIX-Rechner

Gegeben sei der nachfolgende Algorithmus sowie ein Ausschnitt der MMIX-Code-Tabelle (Bild 6.1), eines Register- (Bild 6.2) sowie eines Datenspeichers (Bild 6.3 nächste Seite):

	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*

Algorithmus (Dezimal): $\frac{a}{(b-10c)^2+1024}$

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x86	0x00 00 00 00 00 00 62 10	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 09	Nicht veränderbar
…	…	

Bild 6.2: *Registerspeicher*

a) 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. Führen Sie den 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 in der Registerzelle mit dem Kommentar *Zwischenergebnis*.

1
2
3
4
5

Punkte

b) Nehmen Sie an, der Inhalt des Datenspeichers entspricht dem Zustand in Bild 6.3. Der Registerspeicher (Bild 6.2 vorherige Seite) entspricht dem Zustand nach der Ausführung des Algorithmus. Laden Sie ein Okta ab Speicherstelle 0x0 … 62 0B in die Variable a im Registerspeicher. Wie lautet der hierfür notwendige Befehl als Assembler-Code? Welchen Wert hat die Variable c nach dem Ladevorgang?

Befehl:

Wert:

Datenspeicher	
Adresse	Wert
…	…
0x00 00 00 00 00 00 62 07	0x54
0x00 00 00 00 00 00 62 08	0x32
0x00 00 00 00 00 00 62 09	0x10
0x00 00 00 00 00 00 62 0A	0xFE
0x00 00 00 00 00 00 62 0B	0xDC
0x00 00 00 00 00 00 62 0C	0xBA
0x00 00 00 00 00 00 62 0D	0x98
0x00 00 00 00 00 00 62 0E	0x76
0x00 00 00 00 00 00 62 0F	0x54
0x00 00 00 00 00 00 62 10	0x32
0x00 00 00 00 00 00 62 11	0x10
…	…

Bild 6.3: *Datenspeicher*

c) Welche Einstellungen haben die 3 Multiplexer bei der Ausführung des Befehls ADDI $0x00 $0x01 0x02?

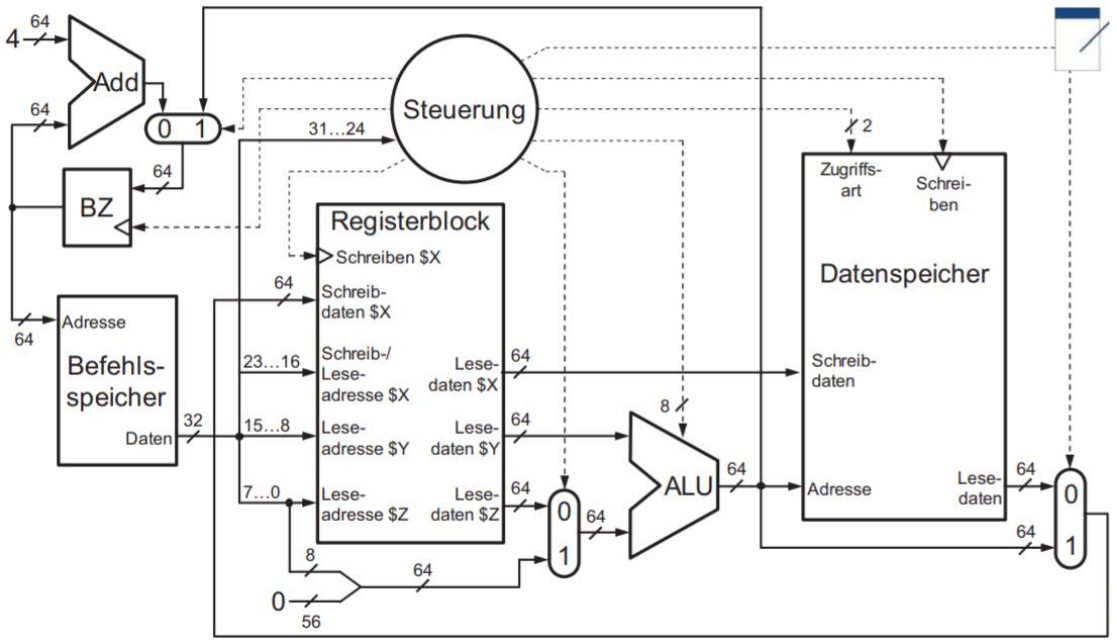


Bild 6.4: *MMIX-Architektur*

	0	1
Multiplexer am Befehlszähler	( )	( )
Multiplexer vor ALU	( )	( )
Multiplexer nach Datenspeicher	( )	( )

Punkte

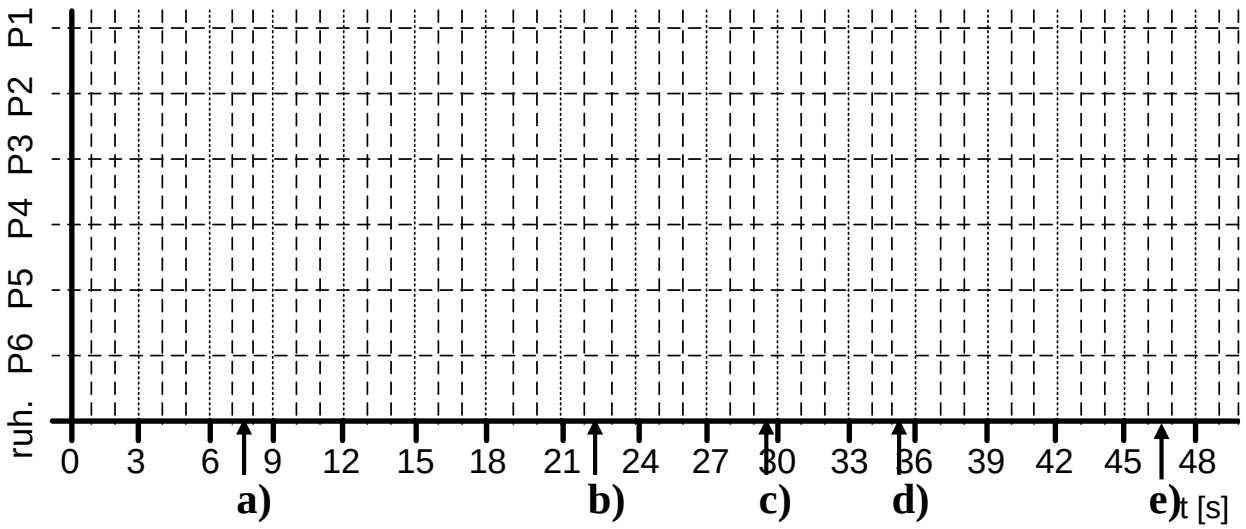
7. Asynchrones Scheduling, präemptiv, Round Robin (RR)

Gegeben seien die folgenden sechs Prozesse (Bild 7.1), welche jeweils ab dem Zeitpunkt *Start* eingeplant werden sollen. Zur Abarbeitung einer Task wird die Rechenzeitspanne *Dauer* benötigt. Periodische Tasks werden mit der Häufigkeit *Frequenz* erneut aufgerufen. Erstellen Sie im untenstehenden Diagramm das präemptive Scheduling nach dem Schema Round-Robin für den Zeitraum $t = [0; 50]$ s für einen Einkernprozessor. Treffen innerhalb eines Zeitschlitzes mehrere Tasks ein, beachten Sie *zuerst LIFO* und *anschließend die Prioritäten*. Ein Zeitschlitz hat eine Größe von drei Sekunden. Kreuzen Sie im Lösungskasten an, welche Tasks zu den gekennzeichneten Zeitpunkten aktiv sind.

	Priorität	Start	Dauer	Frequenz
P1	1 (hoch)	4 s	3 s	13 s
P2	2	2 s	11 s	einmalig
P3	3	0 s	4 s	einmalig

	Priorität	Start	Dauer	Frequenz
P4	4	4 s	2 s	einmalig
P5	5	5 s	3 s	einmalig
P6	6 (niedrig)	23 s	5 s	17 s

Bild 7.1: *Taskspezifikation*



a) P1 ( ), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 ( ), ruhend ( )

b) P1 ( ), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 ( ), ruhend ( )

c) P1 ( ), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 ( ), ruhend ( )

d) P1 ( ), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 ( ), ruhend ( )

e) P1 ( ), P2 ( ), P3 ( ), P4 ( ), P5 ( ), P6 ( ), ruhend ( )

Punkte

8. Semaphoren

Gegeben seien die folgenden drei Tasks T1 bis T3 sowie die dazugehörigen Semaphoren S1 bis S3. Die Startwerte der Semaphoren entnehmen Sie der Hilfstabelle (Bild 8.1). Entwerfen Sie die Semaphorenoperationen derart, so dass der eindeutige Taskablauf $\overline{T3, T1, T2, T1}$ entsteht. Tragen Sie die minimal notwendigen Operationen analog zur exemplarischen Semaphorenzuweisung (Bild 8.2) in die Antworttabelle ein. Beantworten Sie zudem die angegebenen Fragen.

Task	S1	S2	S3
-	0	1	2
T3			
T1			
T2			
T1			
T3			
T1			
T2			
…			

Bild 8.1: *Hilfstabelle Taskablauf*

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

Bild 8.2: *Exemplarische Semaphoren-zuweisung*

T1	T2	T3
…	…	…

Können Semaphoren Werte <0 annehmen?

( ) Ja ( ) Nein

Wie lautet das Phänomen, wenn auf Grund blockierter Semaphoren nieder-priore Tasks vor höherprioren ausgeführt werden?

_____

Punkte

9. Echtzeitbetriebssysteme

Beantworten Sie nachstehende Fragen zu Scheduling mittels PEARL.

Wie lautet in PEARL der Befehl zur Erstellung einer lokalen Semaphore namens *station_frei* mit Startwert 1?

_____

Wie lautet die PEARL-Anweisung, um zyklisch alle 30 Sekunden den Task *gondelstart* zu starten?

_____

In welchem Zustand befindet sich laut „erweitertem Taskzustandsdiagramm von RTOS-UH“ ein PEARL-Task, wenn die Anforderung einer Semaphore fehlschlägt?

_____

In welchem Zustand befindet sich ein Task nach „erweitertem Taskzustandsdiagramm von RTOS-UH“, falls ein ehemals blockierter Task mittels CONTINUE reaktiviert wird, der zyklische Ausführungszeitpunkt jedoch noch nicht eingetreten ist?

_____

10. Bussysteme und Prozessperipherie

Für die Steuerung einer Seilbahn sind Sie mit dem Entwurf des Automatisierungssystems beauftragt. Die Seilbahn verfügt über eine Berg- sowie eine Talstation mit jeweils einer lokalen SPS, welche die zentrale Steuerung der jeweiligen Station durchführt. Jede SPS ist auf kürzestem Weg mit den echtzeitkritischen und sicherheitsrelevanten Sensoren und Aktoren der jeweiligen Station verbunden. Die Kommunikation wird von den beiden SPS initiiert. In der Talstation ist zudem ein SCADA-System (Überwachung + Datensammlung) installiert, das kontinuierlich die beiden SPS per direkter Kabelverbindung überwacht.

Beantworten Sie die nachfolgenden Designentscheidungen zum Systementwurf. *Hinweis: Bei Multiple-Choice Fragen ist jeweils nur eine der Antworten korrekt*

Welche Bustopologie ist für das System am besten geeignet?

_____

Welches Kommunikationsprotokoll eignet sich für die Strecke Sensor zu SPS?

( ) Ethernet ( ) MOST ( ) CAN ( ) USB

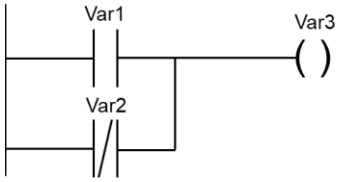
Aufgrund der langen Signalstrecke zwischen Berg- und Talstation kann es zu Taktverschiebungen kommen. Welchen Leitungscode sollten Sie verwenden?

( ) NRZ ( ) Manchester-Code ( ) RZ

Punkte

Aufgabe 11: IEC 61131-3

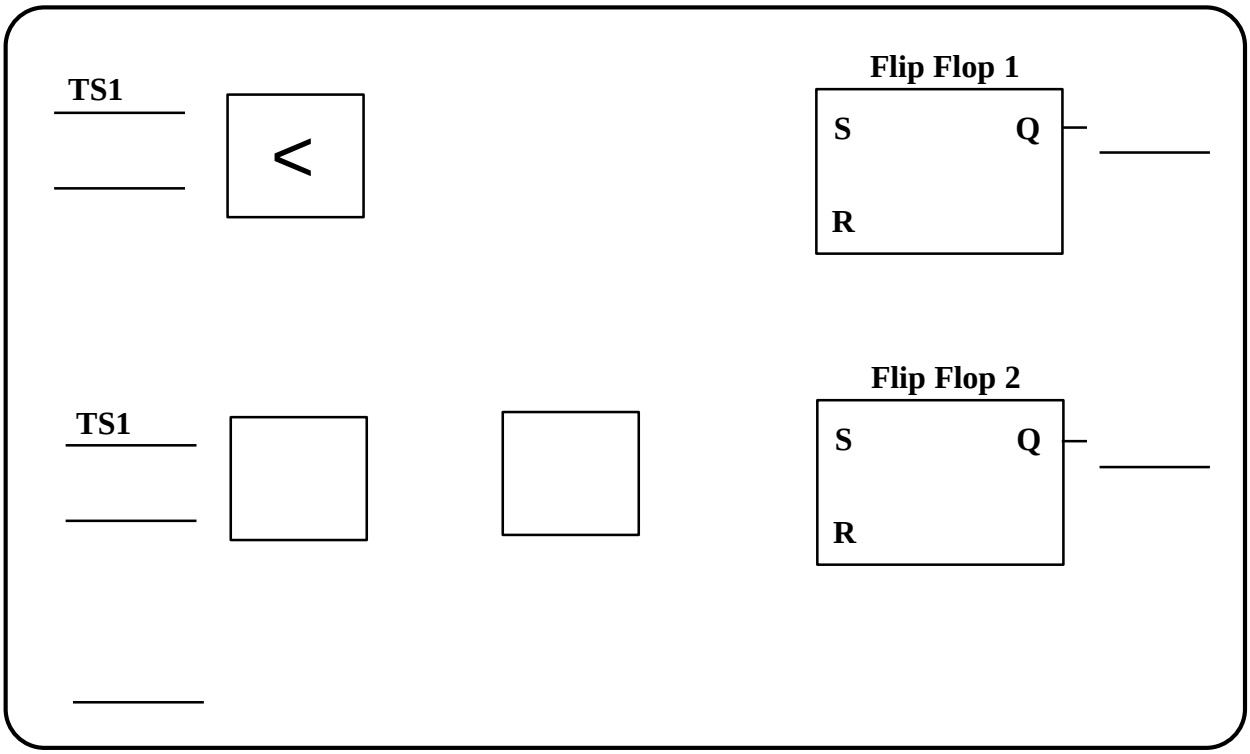
a) Gegeben ist nebenstehender Programmausschnitt in IEC 61131-3-Kontaktplan (KOP).



Überführen Sie den Codeausschnitt in IEC 61131-3-Funktionsbausteinsprache (FBS).

b) Gegeben ist ein Rührkessel zum Erhitzen und Verrühren verschiedener Zutaten:
Der Kessel besitzt zwei Aktoren: Einen Motor *M1* zum Rühren und einen Heizstab *HS1* zum Erhitzen der Zutaten. Für eine optimale Vermischung muss im Rührkessel eine Temperatur zwischen 70°C und 80°C herrschen. Bei einer Temperatur unter 70°C wird der Heizstab angeschaltet. Ab einer Temperatur von 70°C startet der Motor. Übersteigt die Temperatur einen Wert von 80°C, hält der Motor an und der Heizstab wird ausgeschaltet. Die Temperatur wird mit Hilfe eines Temperatursensors *TS1* in °C gemessen.
Wird der Nothalt betätigt (Nothalt = 1), sollen sowohl der Motor als auch der Heizstab unverzüglich ausgeschaltet werden und bleiben.

Vervollständigen Sie im Folgenden das Programm der Reaktorkesselsteuerung in IEC 61131-3-Funktionsbausteinsprache (FBS).
Hinweis: Signalverzögerungen im System sind zu vernachlässigen. Verwenden Sie keine Schaltglieder außer den in der Vorlage bereits vorhandenen. Ergänzen Sie Negationen falls notwendig.



Punkte

Aufgabe 12: Automaten

Gegeben ist folgende Übergangstabelle:

T \ S	s1, 1	s2, 0	s3, 1
a	s2	s1	s2
b	s3	s2	s2

Zeichnen Sie den entsprechenden Übergangsgraphen. Der Startzustand ist s1.

Punkte

Aufgabe 13: Grundlagen C-Programmierung

a) Welche Ausgaben erzeugen die `printf`-Anweisungen in den folgenden Codefragmenten? Wählen Sie die korrekte Antwort (nur Einfachantwort möglich) bzw. füllen Sie die Lücken.

```
double x = 2;
int y = 5;
float f = y / x;
printf("%2.3f", f);
```

( ) 2.5
( ) 2.500
( ) 02.500
( ) 2

```
float x = 8;
int y[] = {6, 12, 18};
int *z = (y + 1);
int **zz = &z;

printf ("%.1f",(y[0] + *--z) / x);
```

→ Ausgabe: _____

```
printf("%i", *(++(*zz)));
```

→ Ausgabe: _____

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

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

( )short ( )float ( )char ( )int
( )int ( )double ( )float ( )char

c) Sie sollen eine Software programmieren, um die beförderte Personenanzahl einer Seilbahn zu erfassen. Hierfür schreiben Sie folgende Werte in eine Datei (bereits geöffnet über Zeiger `handle`)

- einen Zeitstempel `int t`
- gefolgt von dem Kennbuchstaben der Gondel `char x` und
- der beförderten Personenzahl in der Gondel `int y`

Die einzelnen Werte sollen jeweils getrennt durch einen Doppelpunkt (:) und am Ende umgebrochen mittels Zeilenumbruch in die Datei geschrieben werden.
Wählen Sie die korrekte Alternative (nur Einfachantwort möglich) bzw. füllen Sie die Lücke.

```
(1)_____(handle, "(2)_____", t, x, y);
```

(1)
( ) fprintf
( ) print
( ) scanf
( ) printf

(2) _____

Punkte

d) Bestimmen Sie das Ergebnis der Ausdrücke im Dezimalsystem. Gegeben sind die folgenden Variablen:

```
int a = 1;
int b = 9;
int c[] = {10,4,9};
int *d = &c;
```

Nach jedem Ausdruck werden die Variablen nicht (!) auf die oben genannten Werte zurückgesetzt, sondern behalten ihre Werte bei.

c.1)	`(*d++ > b) + (c[1] | a)`	
c.2)	`(*d >> 2 | a) == *(c+2)>>3`	

e) Vervollständigen Sie die Lücken im Lösungskästchen unten.

Eine Endlosschleife soll in jedem Durchlauf auf den Wert von der Variablen `int i` prüfen. Dies geschieht in der Mitte der Schleife, nachdem im oberen Teil der Wert von `i` berechnet wurde (angezeigt durch Kommentar im Lösungskästchen).

- Ist der Wert von `i` kleiner als -1 oder größer als 1000, so soll der Rest dieses Schleifendurchlaufs übersprungen werden.
- Ist der Wert von `i` zwischen 20 und 50 (inklusive beider Werte), so soll die Schleife beendet und verlassen werden.
- Bei jedem anderen Wert von `i` soll die Schleife normal bis zum Ende durchlaufen werden.

```
int i = 0;

while (1)
{
    // Code zum Neuberechnen von i

    ____________________________

    ____________________________

    ____________________________

    ____________________________

    ____________________________

    ____________________________

    ____________________________

    // Rest der Schleife
}
```

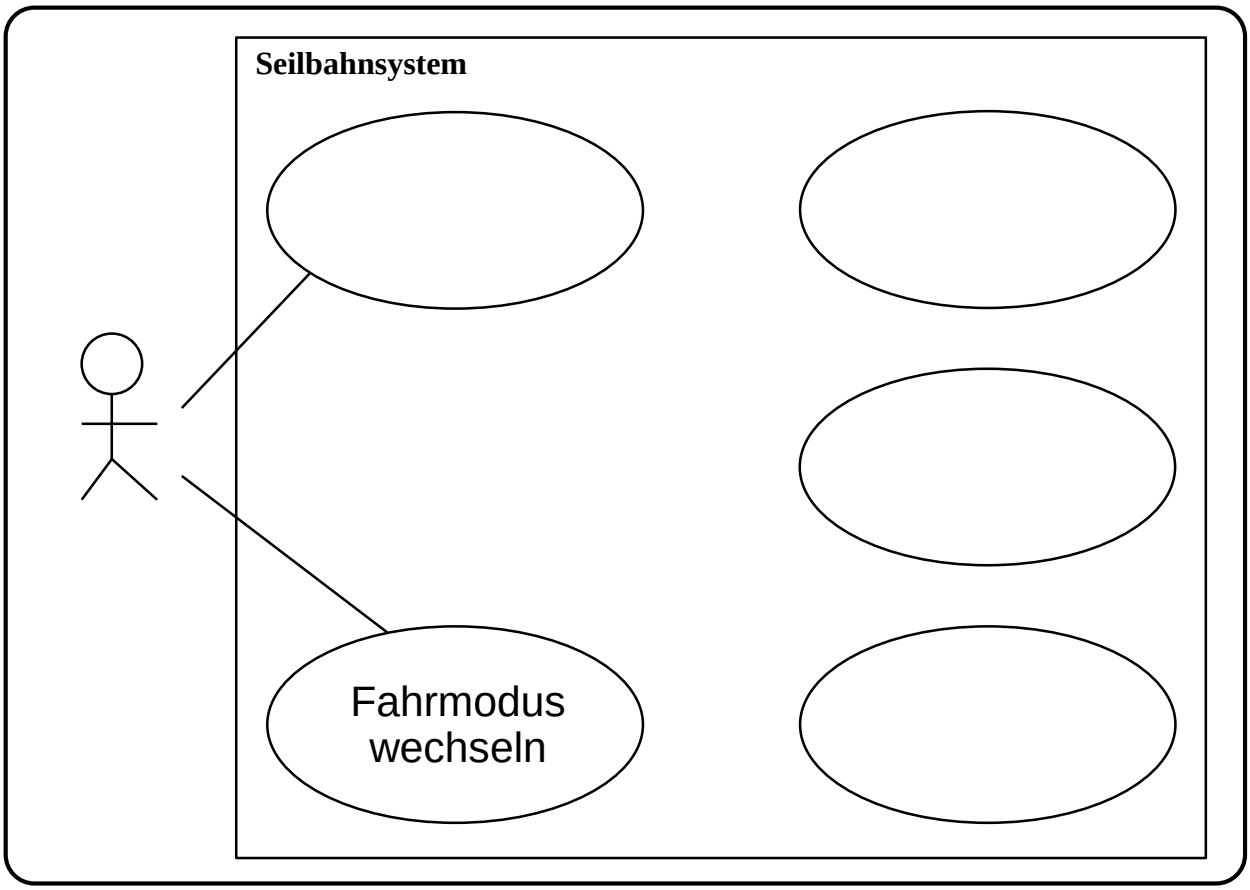
Punkte

Aufgabe 14: Use-Case-Diagramm

Für die nachfolgenden Aufgaben wird ein Seilbahnsystem betrachtet.

Der *Bediener* des Seilbahnsystems kann dieses per Knopfdruck *freigeben*. Bei der Freigabe wird automatisch immer ein *Sicherheitscheck* durchgeführt. Zusätzlich kann sich der Bediener bei der Freigabe vorhandene *Betriebsdaten drucken* lassen.
Das Seilbahnsystem bietet verschiedene Fahrmodi an wie beispielsweise Pendelbetrieb und Umlaufbetrieb. Der Bediener kann zu einem gewünschten *Fahrmodus wechseln*. Dazu muss er sich im Seilbahnsystem *authentifizieren*.

Füllen Sie mithilfe der obigen Angaben das Use-Case-Diagramm der UML aus. Benennen Sie die Akteure sowie die Anwendungsfälle und zeichnen Sie die Beziehungen ein. Bitte achten Sie beim Verbinden der Use-Cases auf die richtige Beziehungsart und –notation.

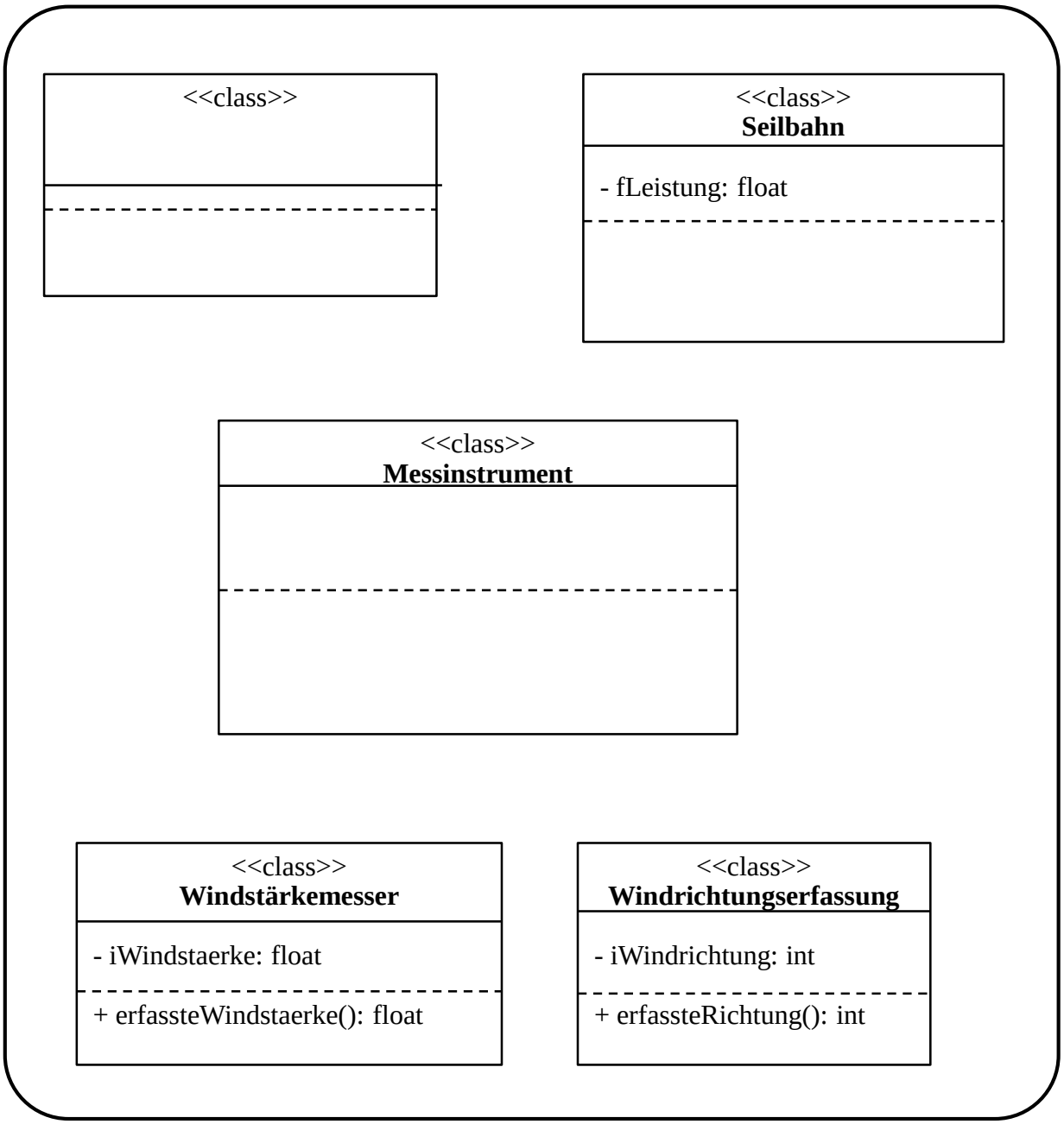


Punkte

Aufgabe 15: Klassendiagramm

Eine Seilbahn betreibt mehrere Gondeln und kann mit den Funktionen „*vorwaerts“* und „*rueckwaerts“* betrieben werden (Rückgabewert bool). Die Tür einer Gondel lässt sich mit der Funktion „*oeffnen*“ sowohl öffnen als auch schließen je nachdem welchen Wert der ganzzahlige Funktionsparameter *„iRichtung“* besitzt.
Außerdem sind an der Gondel beliebig viele Messinstrumente verbaut. Mögliche Messinstrumente sind hierbei der Windstärkemesser und die Windrichtungserfassung. Für jedes Messinstrument wird die *Genauigkeit* als Ganzzahl gespeichert. Die Genauigkeit kann nur über die zugehörige Methode „*hatGenauigkeit*“ abgefragt werden.

Füllen Sie Lücken im folgenden Klassendiagramm anhand der Beschreibung. Lücken **zwischen** Klassen erfordern das Einzeichnen von Beziehungen, Lücken **innerhalb** von Klassen erfordern das Ergänzen von Attributen, Methoden oder Klassennamen. Beachten Sie die Sichtbarkeiten, Kardinalitäten sowie Namenskonventionen.



Punkte

Aufgabe 16: Klassendiagramm zu Code

Sie möchten eine Software zur Steuerung einer Seilbahn schreiben. Bild 16.1 zeigt das vereinfachte UML-Klassendiagramm für die zu implementierende Software.

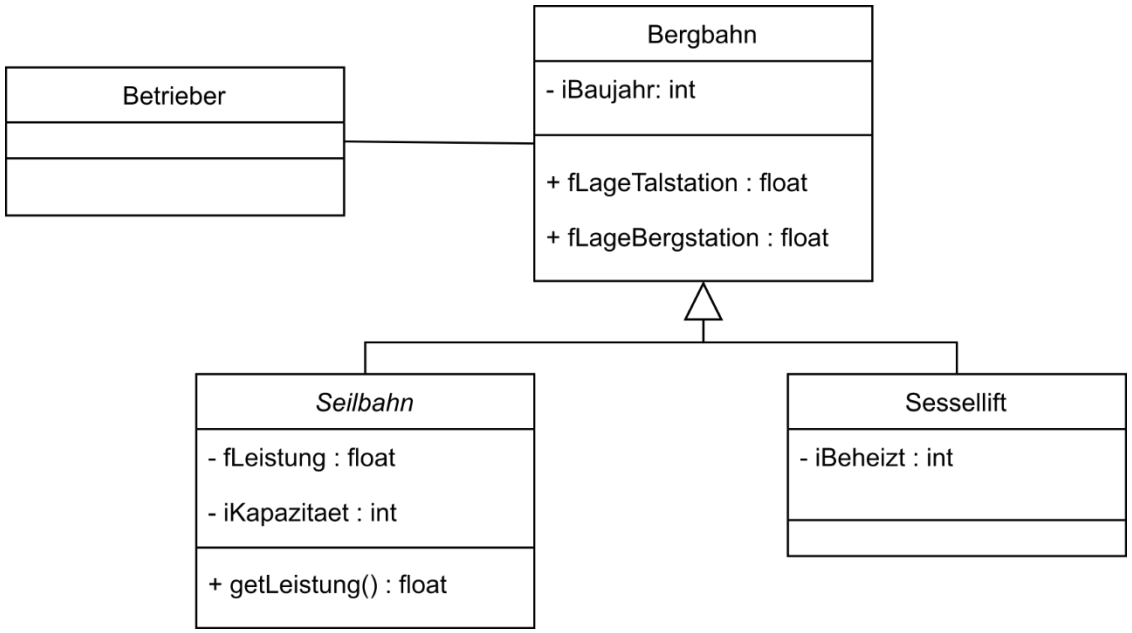


Bild 16.1: Klassendiagramm für Aufgabe 16

Implementieren Sie dieses Klassendiagramm in C++, indem Sie den im Lösungskästchen gegebenen Code ergänzen. Deklarieren Sie Attribute und Methoden. Achten Sie auf die korrekten Sichtbarkeiten und Parameter.

```
class Bergbahn {
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
};

class Seilbahn _____________________________ {
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
};
class Sessellift ___________________________ {
_____________________________________________
_____________________________________________
};
class Betreiber {};
```

Punkte

Aufgabe 17: Zustandsdiagramm

Im Folgenden soll der Betriebsablauf in der Antriebsstation einer Einseil-Pendelbahn (vgl. Bild 17.1) betrachtet werden. Hierbei pendeln zwei fest mit einem Zug- und Tragseil verbundene Gondeln (Gondeln 1 und 2) zwischen zwei Stationen. Eine dieser Stationen, die Antriebsstation, verfügt über einen mit dem Zug- und Tragseil verbundenen Antrieb, welcher über ein Antriebsscheibe und einen Elektromotor das Seil und somit die Gondeln bewegt.

Die Drehrichtung des Motors kann über den Aktor `MRichtung` geändert werden. Über den Aktor `MAn` wird der Motor eingeschaltet und dreht sich dann in die vorgegebene Richtung. Über den Aktor `MHalb`, welcher nur einen Effekt hat wenn der Motor sich dreht, kann in einen Modus mit verringerter Umdrehungsgeschwindigkeit geschalteten werden, um die Gondeln bei deren Ankunft in der Station langsam einfahren lassen zu können.

Jede Ankunftsbucht verfügt über eine Lichtschranke (`LSLinks`, `LSRechts`), welche die Ankunft der jeweiligen Gondel in der Station (Gondel 1 rechts, Gondel 2 links) erkennen kann. Nachdem die Gondel detektiert wurde, soll deren Bewegung über den Motor verlangsamt werden bis diese an dem jeweiligen Endanschlag (`ASLinks`, `ASRechts`) ankommt. Danach wird der Motor ausgeschaltet und dient so gleichzeitig als Haltebremse. Nach der Ankunft soll für drei Sekunden ein Signalton über ein Signalhorn (Ausgang `Horn`) gegeben werden.

Nachdem der Ent- und Beladevorgang abgeschlossen ist, gibt der Bediener per Handtaster (`Start`) eine Freigabe für den Transport. Nach der Freigabe soll zunächst für zwei Sekunden ein Signalton über das Signalhorn (Ausgang `Horn`) ertönen, um die Abfahrt zu signalisieren. Danach soll die Drehrichtung des Motors angepasst und dieser eingeschalten werden, sodass die Gondel wieder zur anderen Station pendelt.

Das System soll unbegrenzt nach den jeweiligen Bedienerfreigaben zwischen den Stationen pendeln (kein Endzustand). Wird das System neu gestartet, soll Gondel 1 zur Antriebsstation gefahren werden, um einen definierten Ausgangszustand herstellen zu können.

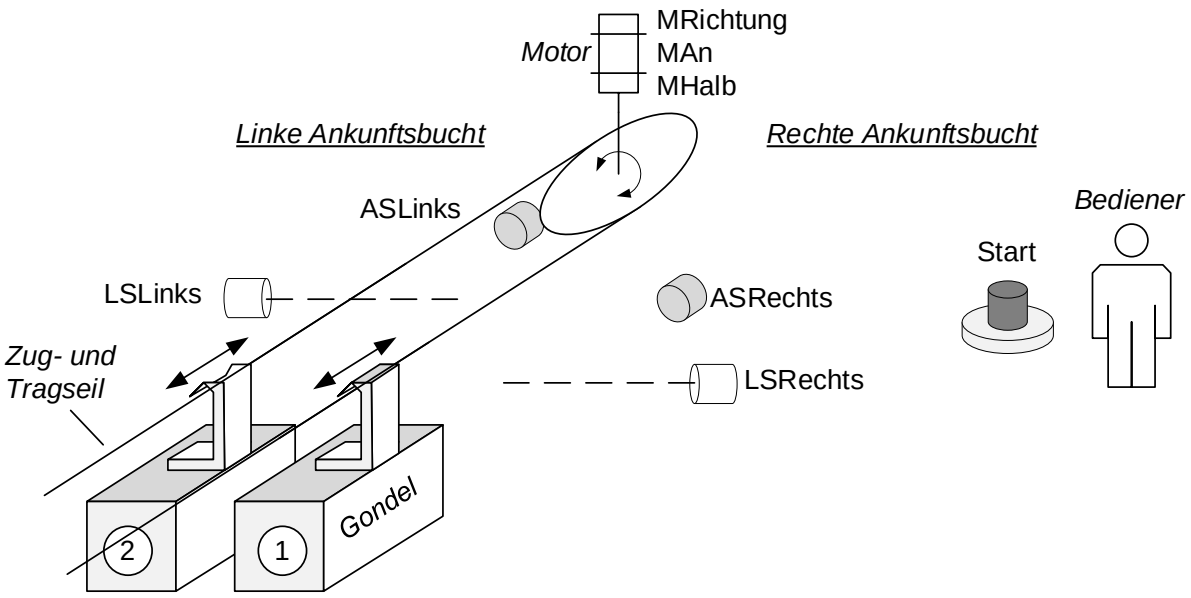


Bild 17.1: Skizze der Antriebsstation der Einseil-Pendelbahn

Punkte

Vorgegeben ist das in Bild 17.2 gezeigte Zustandsdiagramm mit den vorgegebenen Zustandsnummern. Füllen Sie die durch römische Ziffern gekennzeichneten Lücken im Lösungsfeld aus.

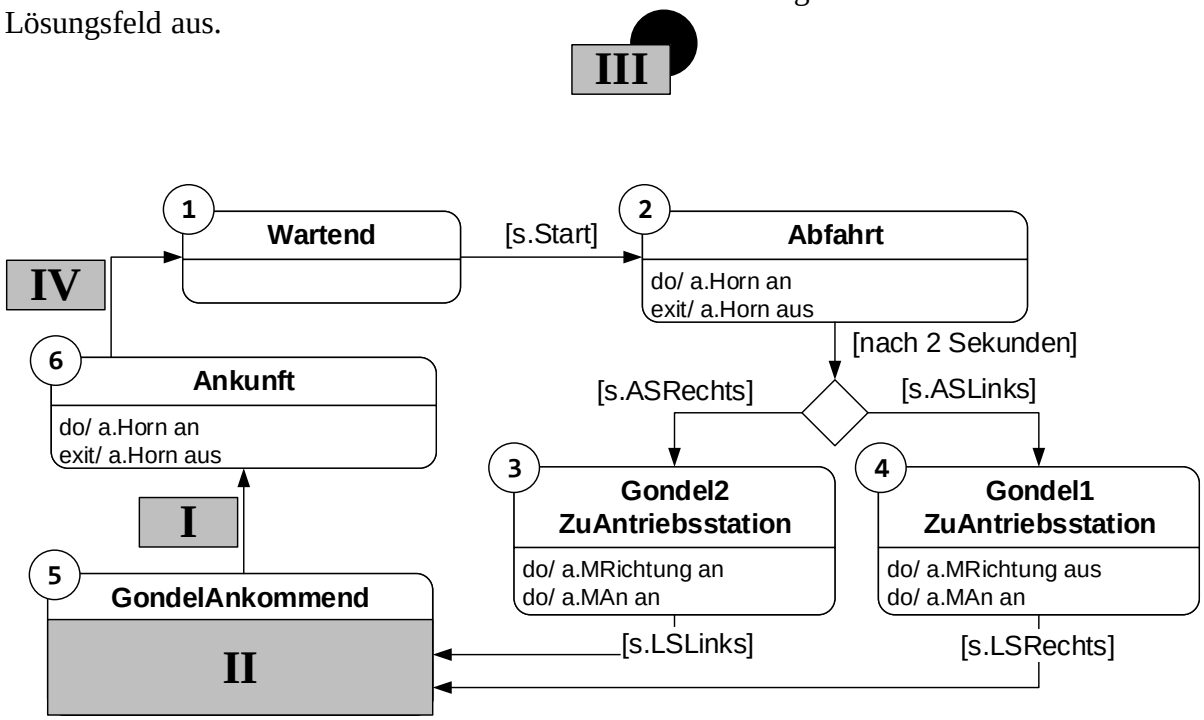
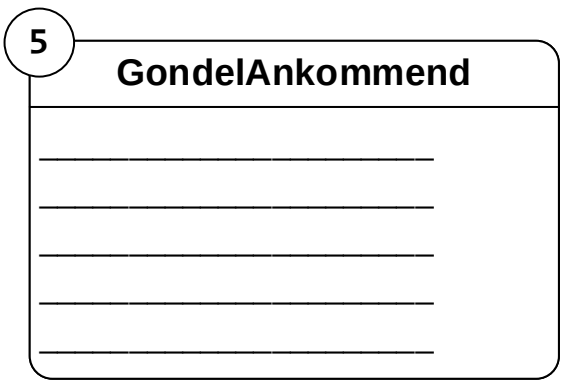


Bild 17.2: Unvollständiges Zustandsdiagramm des Betriebsablaufs

I. Geben Sie die korrekte Wächterbedingung an:

_____

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



III. Geben Sie an, welcher Zustand der Startzustand sein soll:

_____

IV. Geben Sie die korrekte Wächterbedingung an:

_____

Punkte

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 Ein- und Ausgänge sowie erweiterte Variablen zur Verfügung. Weiterhin sind die in Tabelle 18.2 zusammengefassten Funktionen bereits implementiert und erlauben die Interaktion mit der Hardware der Antriebsstation.

Tabelle 18.1: Sensor- und Aktorvariablen der Antriebsstation, sowie erweiterte Variablen

Typ	Name	Beschreibung
AKTOREN	`a.MRichtung`	Beeinflusst die Drehrichtung des Motors. (0) für Gondel 1 zu Station, Gondel 2 weg von Station (1) für Gondel 2 zu Station, Gondel 1 weg von Station
	`a.MAn`	Schaltet die in `a.MRichtung` vorgegebene Rotation des Motors ein (1) oder deaktiviert diese (0).
	`a.MHalb`	Aktiviert (1) oder deaktiviert (0) den Modus mit verringerter Drehgeschwindigkeit des Motors für sanfte Ankunft der Gondeln in der Station. Hat nur einen Effekt wenn der Motor zeitgleich angeschaltet ist.
	`a.Horn`	Aktiviert das Signalhorn (1) oder deaktiviert dieses (0).
SENSOREN	`s.LSRechts`	Zeigt Ankunft der Gondel 1 (rechte Ankunftsbucht) an (1), sonst (0).
	`s.LSLinks`	Zeigt Ankunft der Gondel 2 (linke Ankunftsbucht) an (1), sonst (0).
	`s.ASRechts`	Zeigt Gondel 1 am Endschalter rechts an (1), sonst (0).
	`s.ASLinks`	Zeigt Gondel 2 am Endschalter links an (1), sonst (0).
	`s.Start`	Handtaster zum Starten des Transportvorgangs durch das Bedienpersonal. (1) wenn betätigt, sonst (0).
VARIABLEN	`vplcZeit`	Aktuelle Laufzeit des Programms in ms (positive Ganzzahl)
	`t`	Variable für timer-Programmierung (positive Ganzzahl)
	`schritt`	Aktueller Zustand des Zustandsautomaten (Ganzzahl), welcher ausgeführt wird

Tabelle 18.2: Bereitgestellte Funktionen zur Interaktion mit der Hardware

Funktion	Beschreibung
lSensoren (unsigned int *zeit, SENSOREN *s)	Einlesen der aktuellen Sensorwerte und Zuweisung dieser auf die Variable s. Aktualisiert die Laufzeit des Programms über einen Zeiger auf die Zeitvariable.
sAktoren (AKTOREN *a)	Überträgt die Werte der Aktoren in Variable a an die gesteuerten Ausgänge.

Punkte

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, da diese in anderen Programmteilen so verwendet werden sollen. Zur Interaktion mit der Hardware verwenden Sie die in Tabelle 18.2 gegebenen Funktionen, welche im Header *Seilbahn.h* bereits definiert und implementiert sind. Der Platzhalter `/* ZUSTAENDE */` soll später durch den spezifischen Code der Zustände in Form eines Zustandsautomaten ersetzt werden.

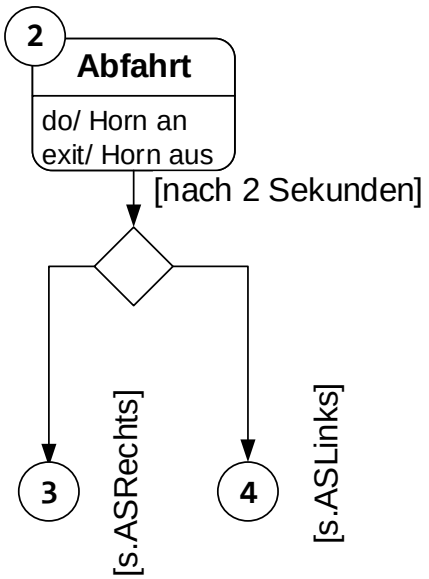
```
_____________________     // Header einbinden

SENSOREN s;
AKTOREN a;
unsigned int t = 0;

_____________________ ()
{  // Deklarationen
   _____________________ // Zeitvariable
   int schritt;          // Schrittvariable

   while(1)              // Zyklische Ausführung
   {
         // Einlesen von Hardware
         ___________________________________;
         ___________________________________
         // Zustandsautomat
         {
                  /* ZUSTAENDE */
         }
          // Schreiben auf Hardware
         ___________________________________

   }
   return 1;      // Wird nicht erreicht
}
```

Punkte

b) Implementierung des Zustands „Abfahrt“

Der Zustand „Abfahrt“ soll im Folgenden implementiert werden. Der Rumpf des Falls (case 2) ist bereits vorgegeben, ergänzen Sie den Code, so dass der Zustand wie im Zustandsdiagramm 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.

```
case 2:   // Abfahrt
```

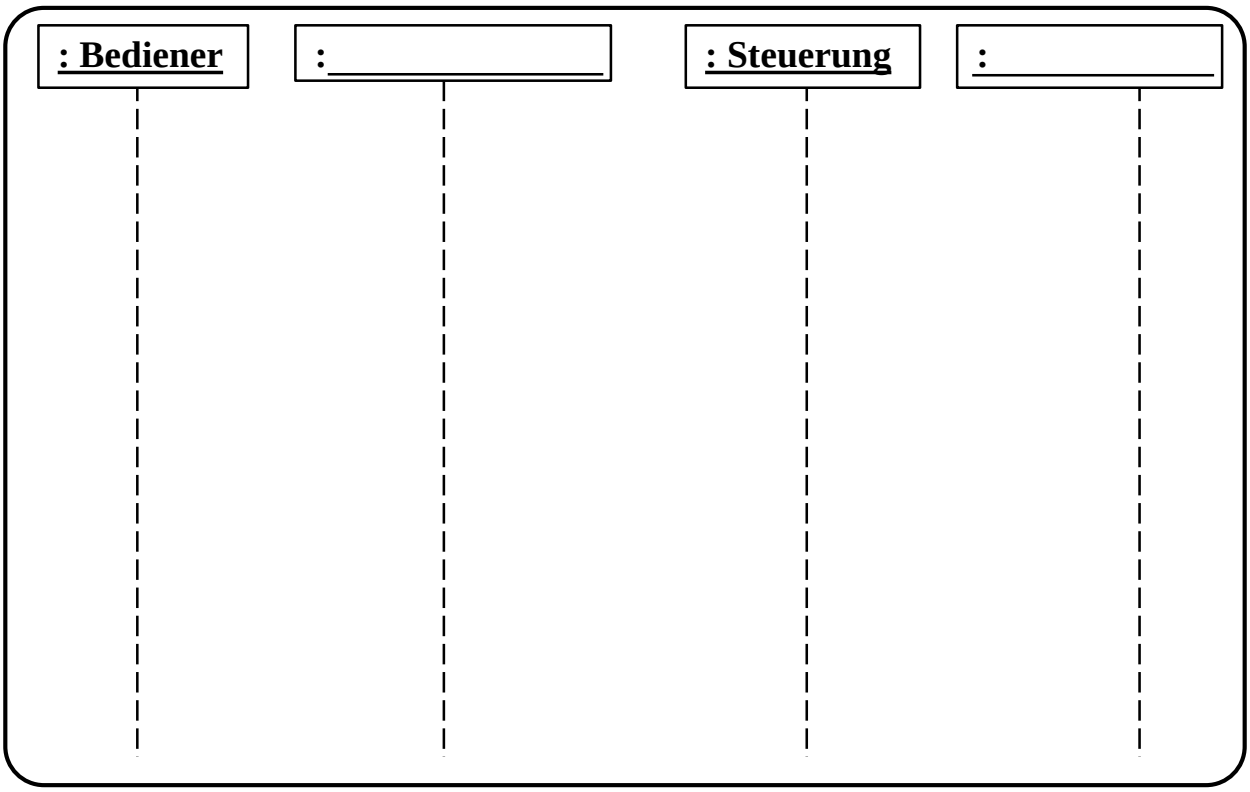


Punkte

Aufgabe 19: Sequenzdiagramm zum Use-Case *Freigeben*

Der Bediener *drückt* den Handtaster für die Freigabe und kann danach weitere Prozesse ausführen. Der Handtaster meldet das Signal an die Steuerung (*meldeFreigabe*) und bleibt daraufhin für maximal 4s aktiv. Währenddessen spielt die Steuerung über das Horn für 2s einen *Signalton* ab. Das Horn meldet der Steuerung zurück, wenn die 2s vorbei sind und es wieder inaktiv ist. Durch einmaliges *aufleuchten* vor Ablauf der 4s benachrichtigt der Handtaster den Bediener über das Ende des Freigabevorgangs.

Ergänzen Sie das untenstehende Sequenzdiagramm entsprechend der Beschreibung.



Punkte

Aufgabe 20: Algorithmen und Datenstrukturen

In dieser Aufgabe soll ein Überwachungssystem realisiert werden, welches Daten zur beförderten Personenzahl während des Betriebs erfassen soll. Weiterhin soll eine einfache Datenbankfunktionalität über Dateien realisiert werden, um historische Passagierzahlen speichern und wiederherstellen zu können.

Die Datenbank soll hierbei in Form einer kommagetrennten Textdatei realisiert werden. Die Erfassung der aktuell beförderten Personenzahl soll **als doppelt verkettete Liste** (bei Definition des LISTENELEMENTS berücksichtigen) umgesetzt werden, **wobei jedes Listenelement für eine besetzte Gondel steht, welche die Station verlässt**.

Gespeichert werden sollen/sind folgende Werte als Listenelemente vom Typ `LISTENELEMENT` und in der Textdatei (in Reihenfolge):

- Aktueller Zeitstempel (`t`) positive Ganzzahl als vorzeichenlose 16-bit Ganzzahl.
- Nummer der Gondel (`GondelNum`), für welche die Daten gelten als 16-bit Ganzzahl.
- Personenzahl in der Gondel (`GondelPers`) als vorzeichenlose 32-bit Ganzzahl.

Im Listenkopf vom Typ `LISTENKOPF` sollen neben der Adresse des ersten `LISTENELEMENTS` folgende Informationen vorgehalten werden:

- Gesamtzahl der bis dahin beförderten Personen (`Befoerdert`) als vorzeichenlose 32-bit Ganzzahl.
- Durchschnittliche Personenzahl pro Gondel während der letzten fünf Abfahrten (`Durchschnitt`) als vorzeichenlose Gleitkommazahl einfacher Genauigkeit.

a) Definition von Listenkopf und Listenelement

Vervollständigen Sie in den folgenden Lösungskästchen gemäß Angabe die Definition des Listenelements sowie des Listenkopfes für die Realisierung der doppelt verketteten Liste. Achten Sie bei der Auswahl der Datentypen auf möglichst hohe Speichereffizienz.

```
_____________________ LISTENELEMENT {
    struct _____________________ pNext;
    struct _____________________ pPrev;
    ___________________________________
    ___________________________________
    ___________________________________
    ___________________________________
} LISTENELEMENT;
```

```
_____________________________ {
    ___________________________________
    ___________________________________
    ___________________________________
    ___________________________________
    ___________________________________
} LISTENKOPF;
```

Punkte

b) Funktion zum Einfügen hinter einem Element der doppelt verketteten Liste

Sie sollen nun eine Funktion `insertAfter` implementieren, welche es erlaubt, ein neues `LISTENELEMENT element` zwischen einem bestimmten Element der Liste und vor einem anderen in die doppelt verkettete Liste einzufügen.

Hierfür wird Ihnen der `LISTENKOPF kopf` der doppelt verketteten Liste, das `LISTENELEMENT prev`, nach dem das neue `LISTENELEMENT` eingefügt werden soll, sowie das neue `LISTENELEMENT element` als Funktionsargumente übergeben.

Nach Hinzufügen des Elements müssen die Statistikdaten im `LISTENKOPF` ggf. neu berechnet werden. Hierfür wird am Funktionsende die Berechnungsfunktion `berechneStatistik` mit dem `LISTENKOPF` als Funktionsargument aufgerufen.

Sie können bei der Implementierung immer davon ausgehen, dass alle Funktionsargumente existieren und korrekt initialisiert sind (die Funktion soll nie ein `LISTENELEMENT` als letztes der Liste einfügen).

```
void insertAfter (LISTENKOPF* kopf,
LISTENELEMENT* prev, LISTENELEMENT* element)
{
    _____________________________________
    _____________________________________
    _____________________________________
    _____________________________________
    _____________________________________
    berechneStatistik(kopf);
}
```

Punkte

c) Aktualisieren der Beförderungsstatistik (Funktion `berechneStatistik`)

Sie sollen nun die Funktion `berechneStatistik` implementieren, welche nach jeder Änderung an den Daten zur Aktualisierung der Statistikdaten im `LISTENKOPF` aufgerufen werden muss.

In der Funktion werden zunächst die bestehenden Daten zurückgesetzt (gegeben). Anschließend muss das erste Element der Liste ausgewählt werden und so lange zum nächsten Element der Liste gegangen werden, bis das letzte Element der Liste (Adresse auf `pNext` gleich `NULL`) gefunden wurde. Anschließend muss die Statistik über die letzten fünf Elemente der Liste erneut berechnet werden und im Listenkopf abgespeichert werden.

Vervollständigen Sie die Implementierung der Funktion `berechneStatistik` im nachfolgenden Lösungskästchen. Sie können für die Implementierung annehmen, dass die Liste immer mindestens fünf Elemente beinhaltet (eine Prüfung ist nicht notwendig).

```
void berechneStatistik (LISTENKOPF* kopf)
{
    int i = 0; // Zaehlervariable
    kopf->Befoerdert = 0; // Ruecksetzen
    kopf->Durchschnitt = 0;
    // Erstes Listenelement zwischenspeichern
    LISTENELEMENT* tmp = kopf->pFirst;
    // Letztes Listenelement in Schleife suchen
    // In Schleife Zeiger immer ein Element weiter
    _____________________________________________
    {
    _____________________________________________
    _____________________________________________
    }
    // Statistik für letzte fünf Gondeln berechnen
    _____________________________________________
    {
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    }
    _____________________________________________
    _____________________________________________
    _____________________________________________
    return;
}
```

Punkte

d) Einlesen der Werte aus der Datenbankdatei

Im letzten Schritt sollen die Funktionalität zum Einlesen der historischen Passagierzahlen aus der Datenbankdatei `datenbank.csv` implementiert werden. In der Datei sind zeilenweise die folgenden Werte durch Komma getrennt gespeichert (vgl. Angabe zu Aufgabe 20 für Datentypen):

`Zeitstempel t,Gondelnummer GondelNum,Personenzahl GondelPers`

Das Bild rechts zeigt einen Ausschnitt aus der Datei zur Verdeutlichung:

```
21,1,16
22,2,32
23,1,3
```

Für das Einlesen muss zunächst die Datei im lesenden Modus geöffnet werden. Anschließend soll zeilenweise so lange eingelesen werden, bis das Dateiende erreicht wurde. Pro Zeile soll dynamisch der Speicher für ein neues LISTENELEMENT reserviert, sowie die Werte aus der Datei eingelesen und dem LISTENELEMENT zugewiesen werden. Eine bereits implementierte Funktion

`insertLast(LISTENKOPF* kopf, LISTENELEMENT* tmp)`

dient dazu, das neu eingelesen Element `tmp` am Ende der Liste (repräsentiert durch den Zeiger auf den `LISTENKOPF kopf`) zu speichern und die Statistik automatisch neu zu berechnen. Diese Funktion ist bereits gegeben und funktionsfähig.

Beachten sie, dass Sie die Datei am Ende wieder schließen müssen. Weiterhin sind keine Fehlerbehandlungen notwendig.

Vervollständigen Sie die Realisierung der Einlesefunktionalität im Lösungskästchen. Deklarieren Sie das hinzuzufügende Element mit dem Namen `tmp`, damit die vorgegebene Initalisierung der Zeigeradressen und das Hinzufügen des Elements korrekt funktioniert.

```
int main() {/* Initialisierung (nicht gezeigt)*/
    // Oeffnen der Datei
    ____________________________________________
    // Start des zeilenweisen Einlesens in Schleife
    ____________________ // Ende nicht erreicht
    {
      ____________________________________________
      // Speicherreservierung
      ____________________________________________
      ____________________________________________
      tmp->pPrev = NULL; // Zeigeradressen inital.
      tmp->pNext = NULL;
      // Einlesevorgang
      ____________________________________________
      ____________________________________________
      ____________________________________________
      insertLast(kopf,tmp); // Hinzufuegen z. Liste
    }
    ____________________ // Datei schließen
    /* Anderer Code (nicht gezeigt) */
}
```