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	
21	
∑	

Prüfung – Informationstechnik

Wintersemester 2019/2020

06.03.2020

Bitte legen Sie Ihren Lichtbildausweis bereit.

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält **29** nummerierte Seiten inkl. Deckblatt.

Bitte prüfen Sie die Vollständigkeit Ihres Exemplars!

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

Aufgaben GL: Grundlagen

Aufgaben GL: 46 Punkte

1. Umrechnung zwischen Zahlensystemen

Über welche Adresslänge verfügt ein im Dualsystem operierender Mikroprozessor mindestens, wenn er rund 65 kByte Speicherblöcke à 8 Byte adressieren kann?

13

2. IEEE 754 Gleitkommazahlen

Rechnen Sie die Dezimalzahl $(7{,}5625)_{10}$ in eine Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) mit folgender Formatierung um:

V e (3 Bit) M (4 Bit)

Vorzeichen: **V = 0**

Mantisse (Binärzahl und normalisiert)

M_2= $(111{,}1001)_2 = (1{,}111001 * 2^2)_2$

Exponent

E = 2

Bias und biased Exponent

B= $2^{(x-1)}-1 = 3$

e = $B + E = (5)_{10} = (101)_2$

Vollständige Gleitkommazahl nach gegebener Formatierung

0 101 1110

Kann die Dezimalzahl +0 bei gegebener Genauigkeit als Binärzahl exakt dargestellt werden?

	Ja	Nein
		X

Was ist die betragsmäßig größte Dezimalzahl, die mit der gegebenen Formatierung dargestellt werden kann?

31

3. Querparität

Folgende Nachricht wurde mittels ungerader Parität gegen Übertragungsfehler geschützt. Ermitteln und markieren Sie die Zeile, die den Übertragungsfehler enthält.

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

4. Logische Schaltungen und Schaltbilder

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

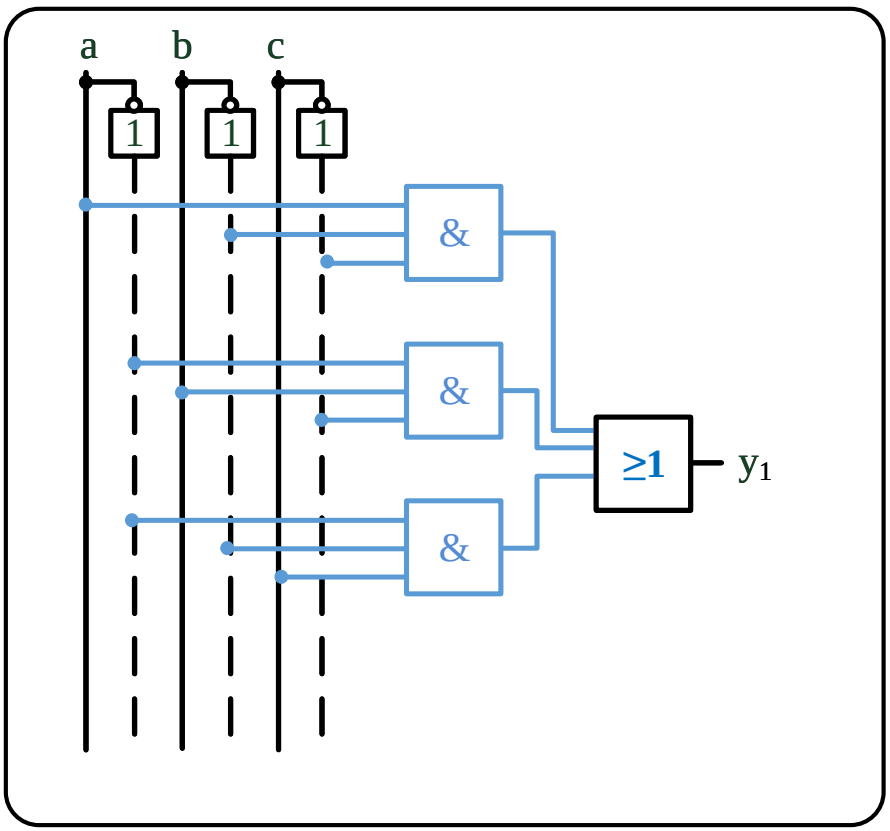


Tabelle 4.1:
Wahrheitstabelle

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

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

( ) Dreikanal Demultiplexer

( ) OR-Gatter für 3 Eingänge

( ) Zweikanal-Multiplexer mit Selektionseingang „b“

(**x**) XOR-Gatter für 3 Eingänge

5. Normalformen und Minimierung

Ermitteln Sie die minimierte DNF für die nebenstehende Wahrheitstabelle (Tabelle 5.1) mittels eines KV-Diagramms.

Hinweis 1: Das Einzeichnen der Schleifen in das KV-Diagramm ist als Lösung ausreichend, die minimierte Formel muss nicht abgelesen werden.

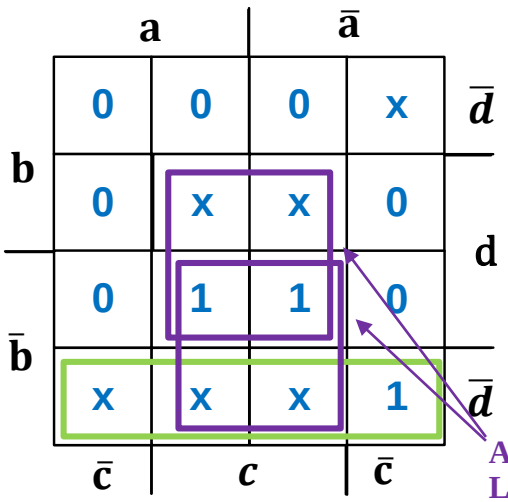
Hinweis 2: Das zweite abgebildete KV-Diagramm dient als Ersatz, falls Sie sich verzeichnen. Kennzeichnen Sie durch Ankreuzen im Feld „dieses KV-Diagramm werten“ eindeutig, welches KV-Diagramm bewertet werden soll.

Hinweis 3: „X“ entspricht „don‘t care“-Einträgen

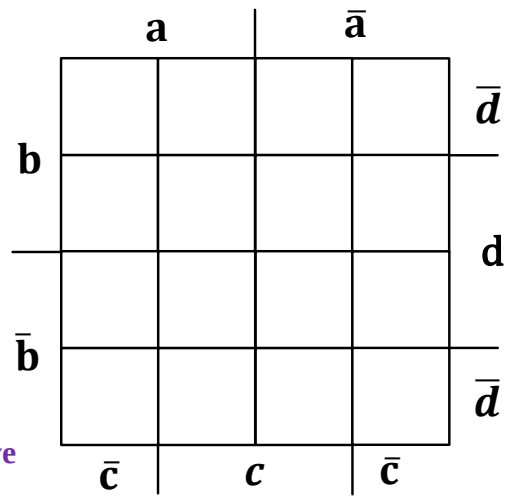
Tabelle 5.1:
Wahrheitstabelle

a	b	c	d	y_1
0	0	0	0	1
0	0	0	1	0
0	0	1	0	X
0	0	1	1	1
0	1	0	0	X
0	1	0	1	0
0	1	1	0	0
0	1	1	1	X
1	0	0	0	X
1	0	0	1	0
1	0	1	0	X
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	X

[X] Dieses KV-Diagramm werten



[ ] Dieses KV-Diagramm werten



6. Flip-Flops

Gegeben ist die folgende Master-Slave-Flip-Flop-Schaltung (Bild 6.1).

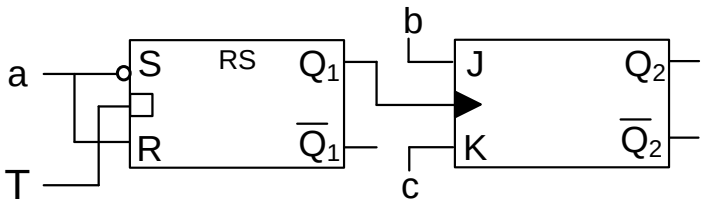
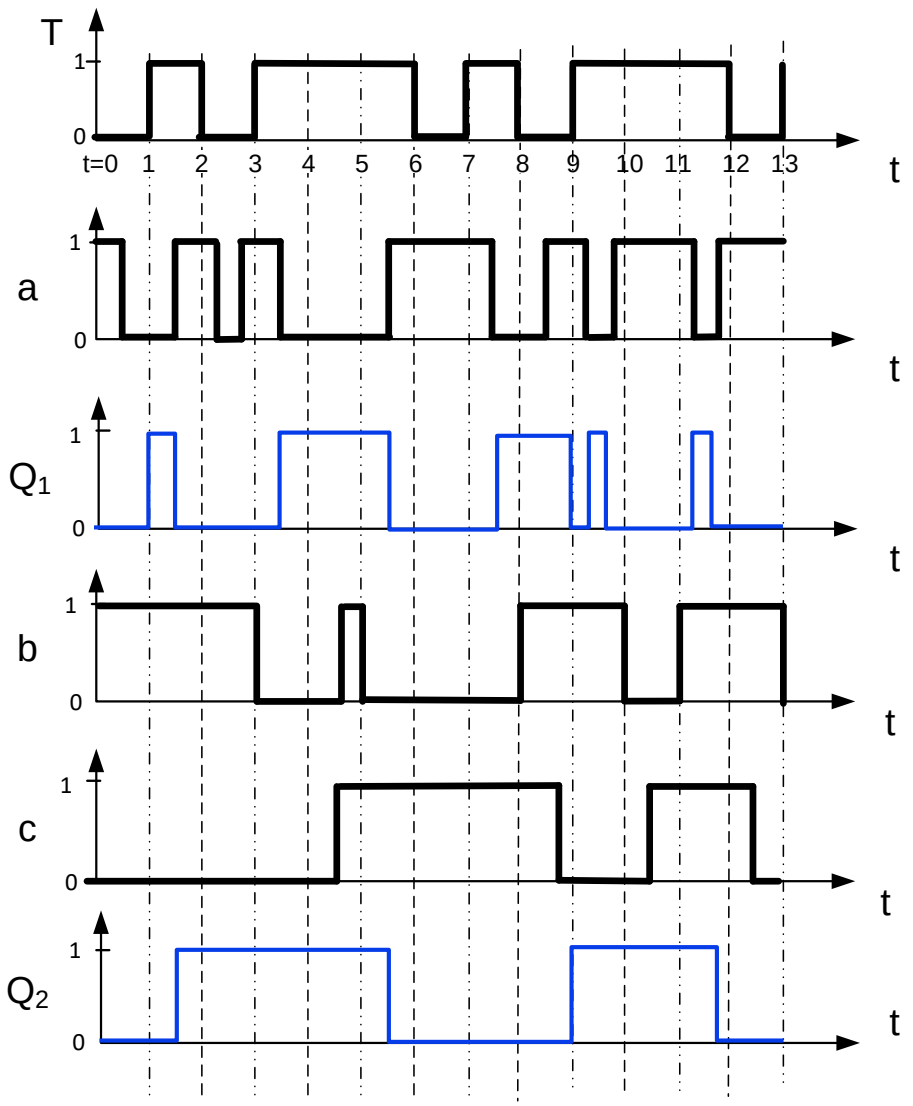


Bild 6.1: *MS-FF*

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.



7. MMIX-Rechner

Gegeben sei der nachfolgende Algorithmus sowie ein Ausschnitt der MMIX-Code-Tabelle (Bild 7.1), eines Register- (Bild 7.2) sowie eines Datenspeichers (Bild 7.3 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 7.1: *MMIX-Code-Tabelle*

Algorithmus: $\frac{b-(266a)^2}{254}$

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x86	0x00 00 00 00 00 00 62 0F	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 08	Zwischenergebnis
$0x8B	0x00 00 00 00 00 00 01 00	Nicht veränderbar
…	…	

Bild 7.2: *Registerspeicher*

a) Im Registerspeicher eines MMIX-Rechners befinden sich zu Beginn die in Bild 7.2 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 7.1 umrahmten Befehlsbereiche. Speichern Sie die Zwischenergebnisse nach jedem Befehl des Algorithmus in der Registerzelle mit dem Kommentar *Zwischenergebnis*.

```
1  ADDI $0x8A $0x8B 0x0A
2  MUL  $0x8A $0x87 $0x8A
3  MUL  $0x8A $0x8A $0x8A
4  SUB  $0x8A $0x88 $0x8A
5  DIVI  $0x8A $0x8A 0xFE
```

oder MUL $0x8A $0x8A $0x87

b) Nehmen Sie an, der Inhalt des Datenspeichers entspricht dem Zustand in Bild 7.3. Der Registerspeicher (Bild 7.2) entspricht dem Zustand nach der Ausführung des Algorithmus. Laden Sie ein Tetra ab Speicherstelle 0x0 … 62 0D in die Variable c im Registerspeicher. Wie lautet der hierfür notwendige Befehl als Assembler-Code? Welchen Wert hat die Variable c nach dem Ladevorgang?

Befehl:

LDTI $0x89 $0x86 0x00

Wert:

0x 00 00 00 00 BA 98 76 54

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 7.3: *Datenspeicher*

c) Wie lautet der Maschinenbefehl 0xE5 01 02 03 in Assembler-Code?

INCMH $0x01 0x0203

Aufgaben BS: Betriebssysteme

Aufgaben BS: 58 Punkte

8. Speicherreservierung

a) Gegeben sei der folgende Speicherzustand eines Datenspeichers. Vervollständigen Sie den Ablauf mittels der Methode Least Recently Used (LRU) in der vorgegebenen Tabelle.

Zeit		0	1	2	3	4
Referenz		-	4	1	3	5
Seite 1		3	3	3	3	3
Seite 2		2	2	2	2	5
Seite 3		4	4	4	4	4
Seite 4		5	5	1	1	1
Stapel	0	3	4	1	3	5
	1	2	3	4	1	3
	2	5	2	3	4	1
	3	4	5	2	2	4

b) Beurteilen Sie die nachfolgenden Aussagen zum Thema Speicherreservierung und Adressumformung auf ihre Korrektheit.

	wahr	**falsch**
Durch virtuellen Speicher kann ein logischer Adressraum auch auf einen wesentlich kleineren, physikalischen Adressraum abgebildet werden.	(X)	( )
Die Größe von Seiten kann beim Verfahren Segmentierung mit Seitenwechsel variieren.	( )	(X)
Least Frequently Used ist ein Verfahren des demand paging.	(X)	( )

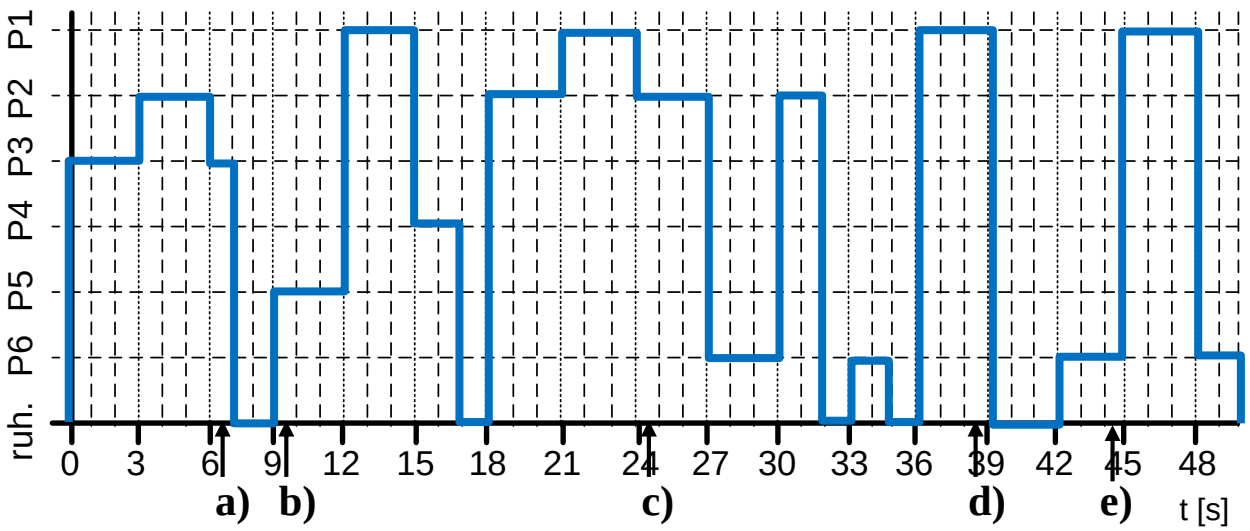
9. Asynchrones Scheduling, präemptiv, Round Robin (RR)

Gegeben seien die folgenden sechs Prozesse (Bild 9.1), welche jeweils ab dem Zeitpunkt *Start* eingeplant werden sollen. Zur Abarbeitung eines Tasks 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 FIFO* 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)	5 s	3 s	13 s
P2	2	1 s	11 s	einmalig
P3	3	0 s	4 s	einmalig

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

Bild 9.1: *Taskspezifikation*



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

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

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

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

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

10. 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 10.1). Entwerfen Sie die Semaphorenoperationen derart, so dass der eindeutige Taskablauf $\overline{T2, T1, T2, T3}$ entsteht. Tragen Sie die minimal notwendigen Operationen analog zur exemplarischen Semaphorenzuweisung (Bild 10.2) in die Antworttabelle ein. Beantworten Sie zudem die angegebene Frage.

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

Bild 10.1: *Hilfstabelle Taskablauf*

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

Bild 10.2: *Exemplarische Semaphoren-zuweisung*

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

Wie heißt ein Zustand, bei dem alle Tasks bedingt durch gesperrte Semaphoren nicht ablauffähig sind?

Deadlock
Alt. Globale Verklemmung

11. Echtzeitbetriebssysteme

Im Folgenden (Bild 11.1) ist ein unvollständiges „erweitertes Taskzustandsdiagramm von RTOS-UH“ gegeben.

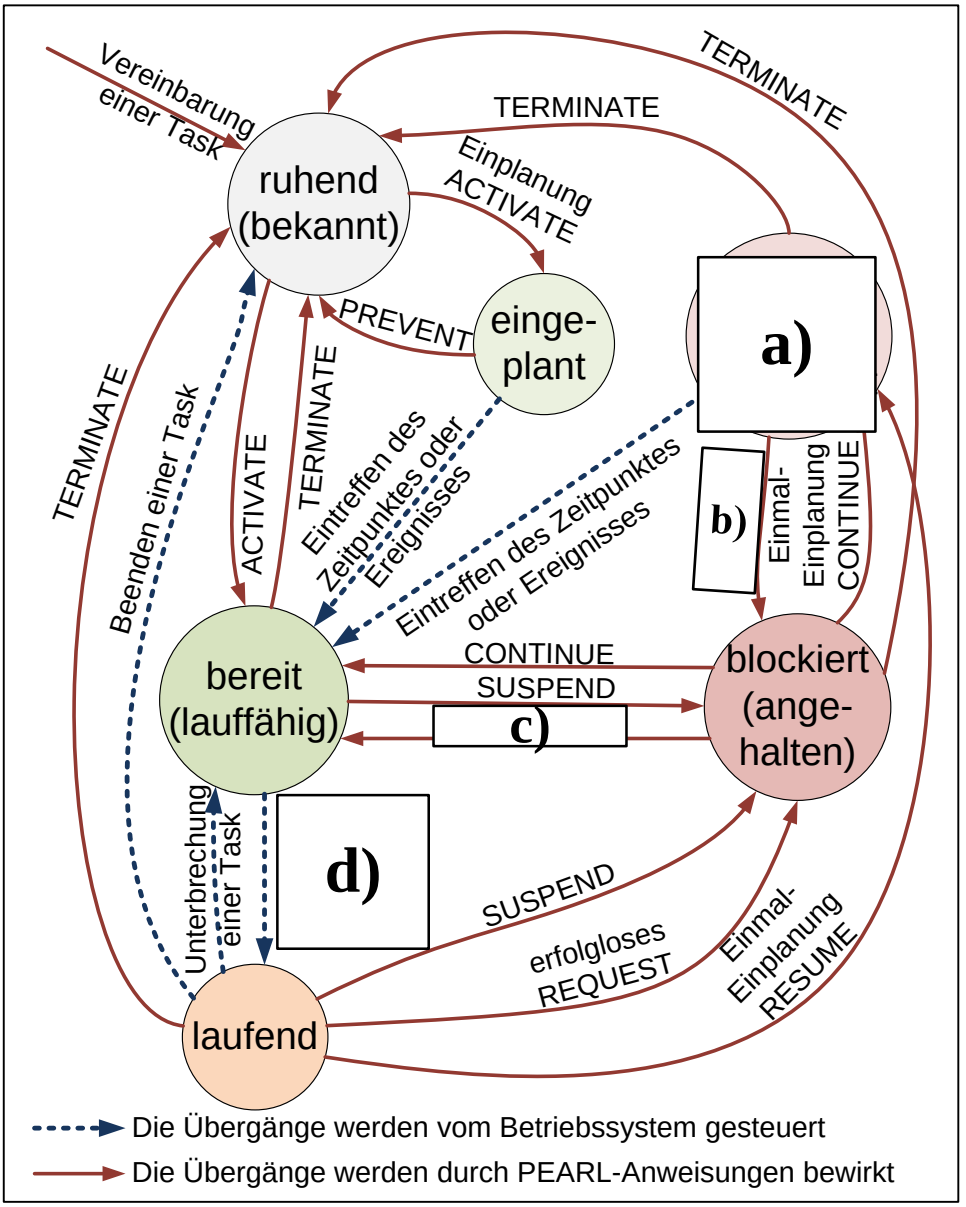


Bild 11.1: *Erweitertes Taskzustandsdiagramm von RTOS-UH*

Bezeichnen Sie die im Diagramm mit Buchstaben markierten Lücken:

a) Blockiert und eingeplant

b) PREVENT

c) RELEASE

d) Starten der Task mit höchster Priorität

12. IEC 61131-3: Funktionsbausteinsprache (FBS)

Ein Reaktorkessel (Bild 12.1) soll über drei Aktoren gesteuert werden: *Vzu* und *Vab* zum Befüllen und Entleeren des Kessels sowie *M1* zur Steuerung der Durchmischung (1 → offen/an, 0 → geschlossen/aus). Der Füllstand wird mit dem analogen Ultraschallsensor *FS1* gemessen (<= 5 → leer, >= 95 → voll).

Ist der Tank leer, soll einmalig der Zulauf geöffnet und der Ablauf geschlossen werden. Ist der Tank nicht leer, soll zudem das Rührwerk *M1* dauerhaft aktiv sein.

Ist der Tank voll, soll der Zulauf einmalig wieder geschlossen werden.

Erhält die Steuerung vom Leitstand das Signal *Nothalt* (Nothalt = 1), soll der Zulauf sofort geschlossen und der Ablauf geöffnet werden. Dieses Verhalten hat Priorität über den Regelbetrieb bis der Nothalt wieder aufgehoben wird.

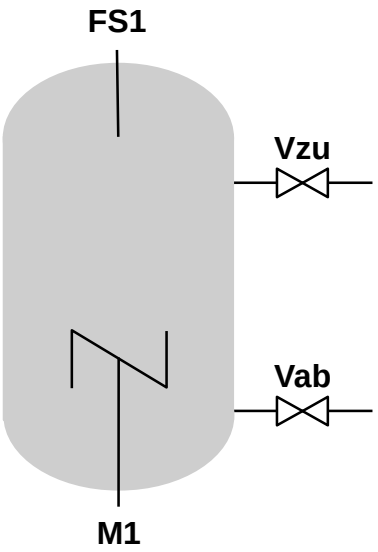
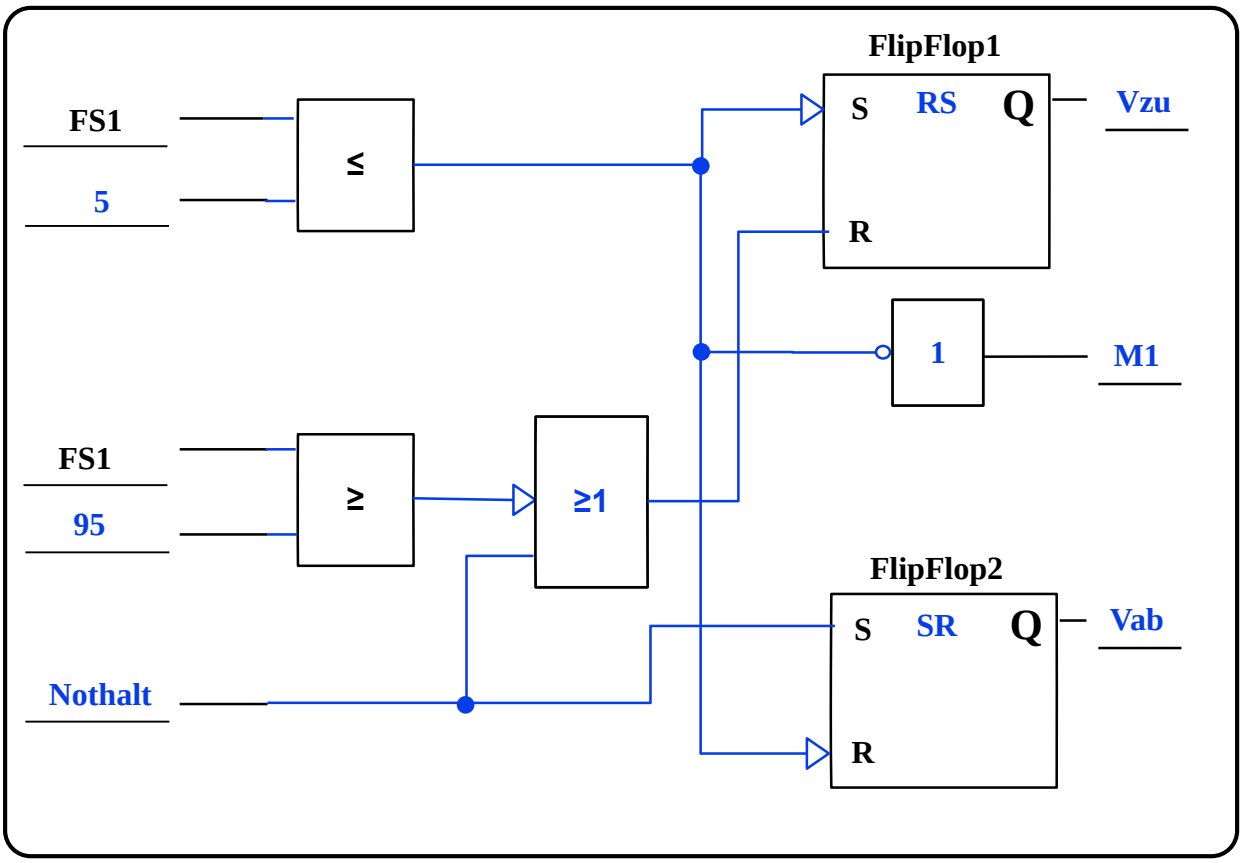


Bild 12.1: *Schematische Darstellung des Reaktorkessels*

Vervollständigen Sie im Folgenden das Programm der Reaktorkesselsteuerung in der Funktionsbausteinsprache.
Hinweis: Signalverzögerungen im System sind zu vernachlässigen. Verwenden Sie keine Schaltglieder außer den in der Vorlage bereits vorhandenen. Ergänzen Sie Negationen oder Flankenerkennungen falls notwendig.



13. Bussysteme

a) Beurteilen Sie die Behauptungen auf ihre Richtigkeit hin. Kreuzen Sie entsprechend an.

	wahr	*falsch*
Bei CAN handelt es sich um ein dezentral gesteuertes Buszugriffsverfahren.	( )	(x)
CSMA/CA eignet sich zum priorisierten Versand wichtiger Nachrichten.	(x)	( )
Der CAN-Bus setzt TDMA als Buszugriffsverfahren ein.	( )	(x)
Bei Profibus-DP handelt es sich um eine Master-Slave-Architektur.	(x)	( )
Firewire eignet sich für sicherheitskritische Anwendungen.	( )	(x)

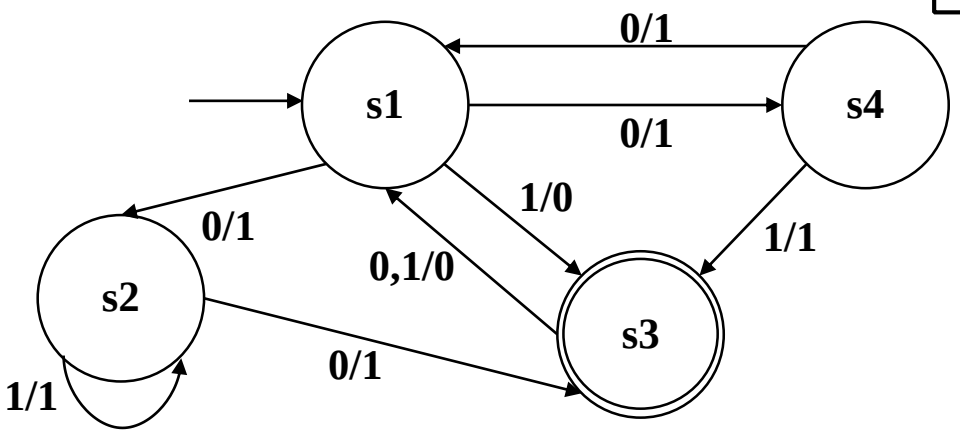
b) Ordnen Sie die nachfolgenden Aufgaben den zuständigen Schichten des ISO 7498-Schichtenmodells zu

	Applikationsschicht	Darstellungsschicht	Sitzungsschicht	Transportschicht	Netzschicht	Sicherungsschicht	Bitübertragungsschicht
Konvertierung von Grey-Code	( )	(x)	( )	( )	( )	( )	( )
Verwendung von NRZ-Code	( )	( )	( )	( )	( )	( )	(x)
Definition von Paritätsbits	( )	( )	( )	( )	( )	(x)	( )

14. Automaten

Aufgabe 14: 14 Punkte

Gegeben ist der folgende Automat.



a) Um welchen Automaten-Typen handelt es sich hierbei?

Mealy

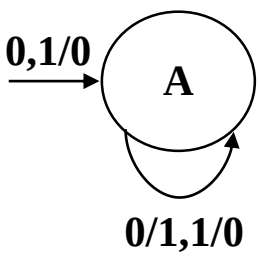
b) Ist der gegebene Automat deterministisch? Woran können Sie dies erkennen?

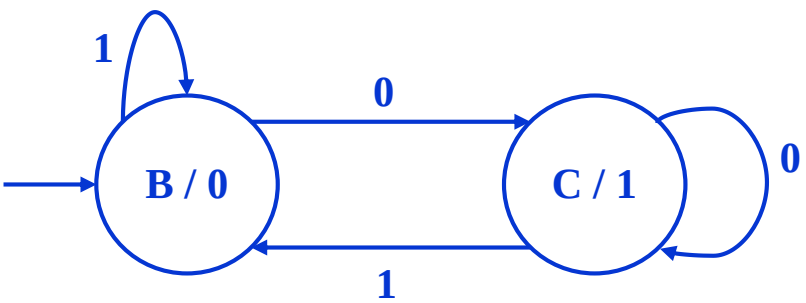
Nein, von Zustand S1 gibt es zwei Folgezustände für Eingang 0.

c) Füllen Sie die Lücken in der folgenden Übergangstabelle. Ignorieren Sie Felder mit „X“.

T	s1	s2	s3	s4
0	X	**s3,1**	s1,0	**s1,1**
1	**s3,0**	s2,1	**s1,0**	X

d) Wandeln Sie den links abgebildeten Automat um (Mealy in Moore, Moore in Mealy).





Aufgabe 15:
15 Punkte

15. C: Grundlagen

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

```
int x = 4;
int y = 2;
float f = y / x;
printf("%.2f", f);
```

( ) 0.5
( ) 0
(x) 0.00
( ) 1

```
float x = 8;
int y[] = {6, 12, 18};
int *z = (y + 1);
float f = *z / x;
printf("%.1f", f);
```

→ Ausgabe: 1.5

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

→ Ausgabe: 18

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 - char * short`

`int + float * long`

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

c) Sie sollen eine Software programmieren, um Windkraftanlagen zu verwalten. Zunächst sollen Informationen über die einzelnen Windkraftanlagen ausgegeben werden. Die Variablen `float h` und `float l` speichern Höhe und Leistung der Anlage.

Sie möchten die Höhe als Ganzzahl in Metern (m) und die Leistung mit 2 Nachkommastellen in Megawatt (MW) ausgeben (Beispielausgabe: `82m, 2000.23MW`). Wählen Sie die korrekte Alternative (nur Einfachantwort möglich) bzw. füllen Sie die Lücke.

```
(1)_________("_____(2)_____", (int) h, l);
```

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

(2) %im, %.2fMW

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

```
int a = 2;
int b = 4;
int c = 5;
int *d = &a;
```

Nach jedem Ausdruck werden die Variablen auf die oben genannten Werte zurückgesetzt.

d.1)	`c + (((a & 2) | b) << a)`	**29**
d.2)	`(b >= *d) + (*d + 1 >= c)`	**1**

e) Füllen Sie die Lücken in den folgenden Codebeispielen, so dass die Aussagen zutreffen. Die Variablen `int i` und `int j` sind bereits definiert.

```
if (_____i >= 2 && i <= 5_____) {
    printf("i ist mindestens 2 und hoechstens 5\n");
}
if (_____j % i != 0_____) {
    printf("j ist kein ganzzahliges Vielfaches von i\n");
}
```

f) Füllen Sie die Lücken im folgenden Lösungskästchen entsprechend der Anweisungen. **Die leeren Zeilen lassen keine Rückschlüsse auf die Länge der korrekten Lösung zu.**

Erstellen Sie eine Schleife, die genau 15-mal durchlaufen wird. Verwenden Sie als Schleifenzähler die Variable `i` und geben Sie diese bei jedem Durchlauf aus.

```
int i;

___for___ (____i = 0; i < 15; i++____)
{
    __printf__ (____"%i", i____);

    ____________________________
}
```

Aufgabe 16:
8 Punkte

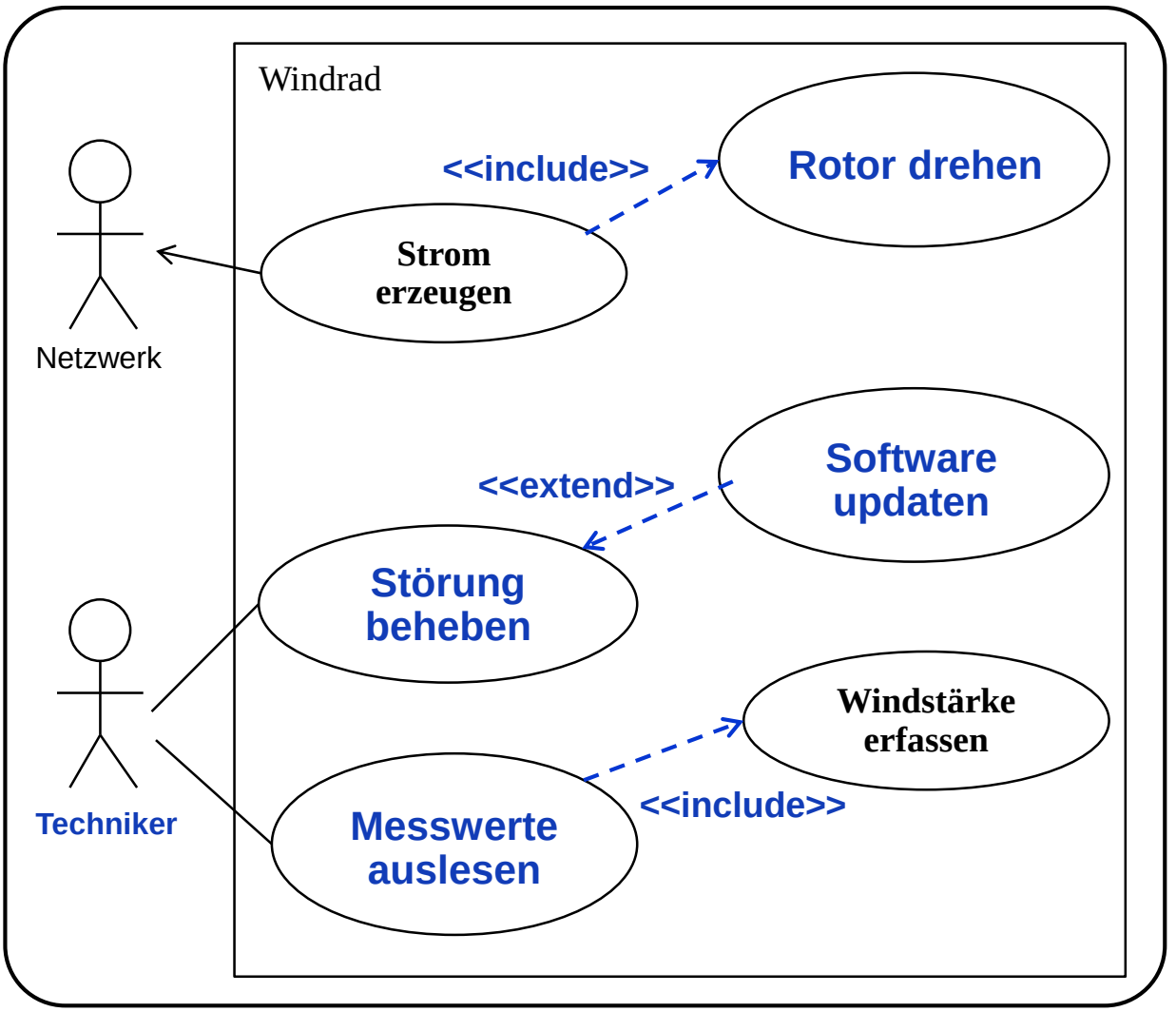
16. Use-Case-Diagramm

Für die folgenden Aufgaben wird ein Windrad einer Windkraftanlage betrachtet.

Mithilfe des Windrads kann Strom für das Netzwerk erzeugt werden. Der Strom wird durch die Funktion „*Rotor drehen*“ des Windrads erzeugt.

Das Windrad wird durch einen Techniker gewartet. Er kann *Störungen beheben* und *Messwerte auslesen*. Je nach Störungsart kann *Software updaten* ausgeführt werden, um die Störung zu beheben. Um stets die aktuellsten Werte zu erhalten, wird immer die *Windstärke erfasst*, sobald der Techniker die Messwerte auslesen möchte.

Füllen Sie mithilfe der obigen Angaben das Use-Case-Diagramm der UML für das Windrad aus. Benennen Sie die Akteure sowie die Anwendungsfälle. Bitte achten Sie beim Verbinden der Use-Cases auf die richtigen Beziehungen.

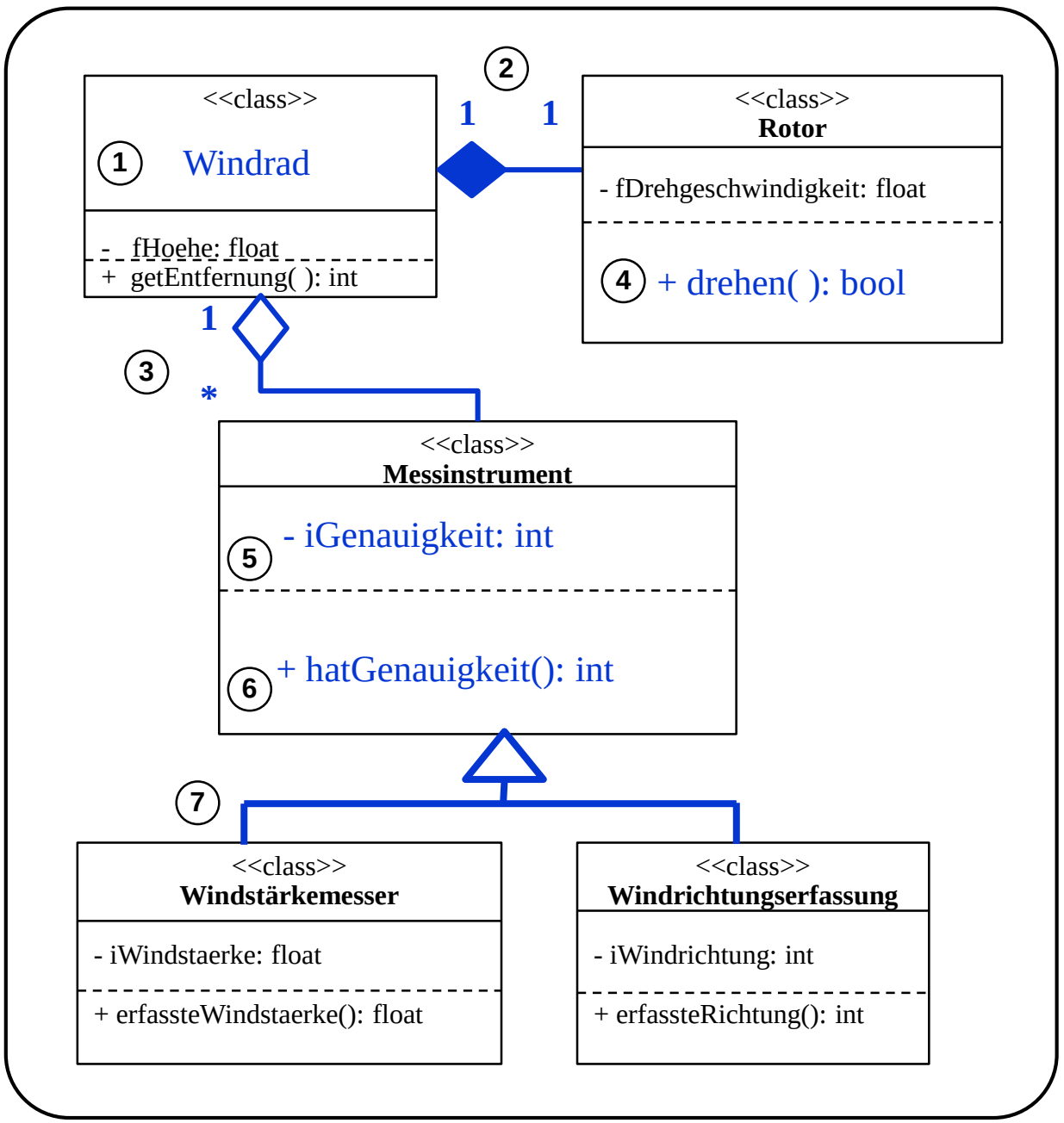


Aufgabe 17:
10 Punkte

17. Klassendiagramm

Ein Windrad besteht per Definition immer aus genau einem Rotor. Der Rotor wird mit der Funktion „*drehen*“ betrieben (Rückgabewert bool). Zusätzlich kann es beliebig viele Messinstrumente beinhalten. 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 die mit Zahlen markierten Lücken im folgenden Klassendiagramm. 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.



Aufgabe 18:
16 Punkte

18. Objektorientierte Programmierung

a) Folgend werden grundlegende Konzepte der objektorientierten Programmierung abgefragt. Bitte wählen Sie aus den Antwortalternativen die korrekte Alternative aus (nur Einfachnennung möglich), oder füllen Sie die markierten Lücken mit dem korrekten Begriff.

1. Ein Attribut soll nur für Instanzen von abgeleiteten Klassen zugänglich sein. Welches Schlüsselwort verwenden Sie?

protected

2. Was beschreibt das Prinzip der Datenkapselung?
( ) Der Zustand eines Objekts kann nicht verändert werden.
( ) Alle Zustandsdaten werden in einem eigenen Objekt zusammengefasst.
(x) Der Zustand eines Objektes kann von außen nur über Methoden verändert werden.
( ) Es kann nicht auf den Zustand eines Objekts zugegriffen werden.

3. Instanzen welcher Klassen können auf private Attribute zugreifen?
( ) Instanzen von Unterklassen der Klasse.
( ) Keine.
(x) Instanzen der Klasse in der das Attribut definiert wurde.
( ) Instanzen von assoziierten Klassen.

4. Was beschreiben die Methoden eines Objekts?
( ) Kommunikation
(x) Verhalten
( ) Zustand
( ) Abstraktion

5. Wann wird ein Konstruktor aufgerufen?
(x) Bei der Erzeugung der Instanz.
( ) Bei der Zerstörung der Instanz.
( ) Bei jeder Verwendung der Instanz.
( ) Bei erstmaliger Verwendung der Instanz.

6. Wie bezeichnen Sie eine lose Beziehung zwischen zwei Klassen?
( ) Aggregation
( ) Vererbung
( ) Komposition
(x) Assoziation

b) Sie möchten eine Software zur Verwaltung von Kraftwerken schreiben. Bild 18.1 zeigt das vereinfachte UML-Klassendiagramm für die zu implementierende Software.

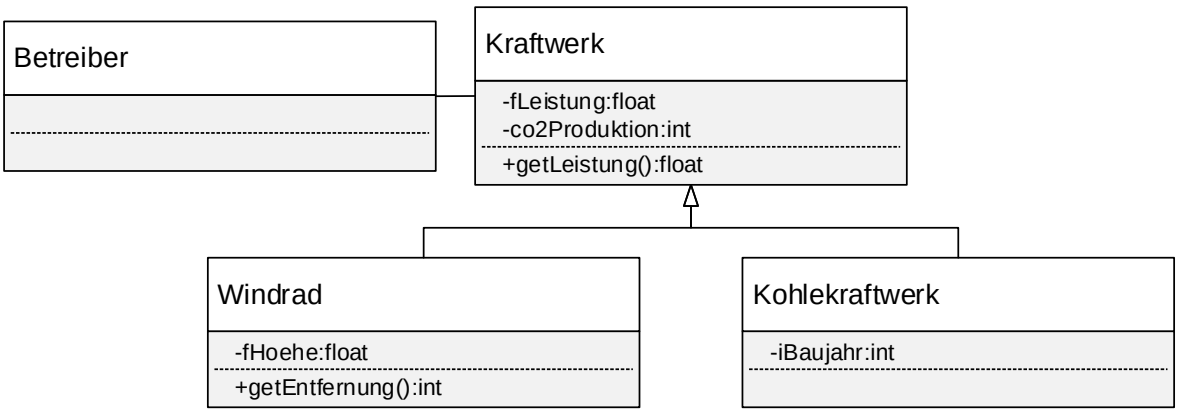


Bild 18.1: Klassendiagramm für Aufgabe 18 b)

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 Kraftwerk {
private:
  float fLeistung;
  int co2Produktion;
  Betreiber* betreiber;
public:
  float getLeistung();

};

class Windrad : public Kraftwerk {
private:
  float fHoehe;
public:
  int getEntfernung();
};
class Kohlekraftwerk : public Kraftwerk {
private:
  int iBaujahr;
};

class Betreiber {};
```

19. Zustandsdiagramm

Aufgabe 19:
8 Punkte

Im Folgenden betrachten wir eine Windkraftanlage (vgl. Bild 19.1). Im Normalzustand befindet sich die Anlage zum Wind ausgerichtet und produziert Strom. Im Betrieb wird kontinuierlich die `Abweichung` zwischen aktueller Windrichtung und Ausrichtung der Windkraftanlage in Grad (°) bestimmt. Sobald die Abweichung mehr als 10° beträgt, soll das Maschinenhaus neu ausgerichtet werden. Hierfür ist zwischen Turm und Maschinenhaus ein Motor installiert, welcher das Maschinenhaus nach rechts (`DrehungR`, bei positiver Abweichung, Zustand `DrehenRechts`) oder links (`DrehungL`, bei negativer Abweichung, Zustand `DrehenLinks`) schwenken kann. Sobald die Abweichung weniger als 3° beträgt, wird in den Normalzustand gewechselt und die Neuausrichtung stoppt.

Weiterhin sind für den Fall einer Überlastung zwei unabhängige Bremssysteme installiert. Zum einen kann der Neigungswinkel (Pitch) der Rotoren zwischen den zwei Stufen „Nullstellung“ und „Maximales aus dem Wind drehen“ umgeschaltet werden. Dies wird durch Aktoren in der Nabe bewerkstelligt, welche durch den Aktor `PitchMax` das „Maximale aus dem Wind drehen“ aktivieren können. Zum anderen ist ein mechanisches Bremssystem an der Rotorachse installiert, welches zusätzlich aktiviert werden kann.
Eine Überlastung tritt auf, wenn die Umdrehungszahl der Achse (`Umdrehungen`) größer als 15 Umdrehungen pro Minute ist und diese Belastung länger als 30 Sekunden anhält. In diesem Fall soll der Neigungswinkel der Rotorblätter auf „Maximales aus dem Wind drehen“ gestellt werden. Ist die Umdrehungsgeschwindigkeit unter 15 Umdrehungen gesunken, wird der Neigungswinkel wieder auf „Nullstellung“ zurückgesetzt und in den Normalzustand gewechselt. Ist jedoch die Umdrehungsgeschwindigkeit auch nach weiteren 10 Sekunden noch größer als 18 Umdrehungen pro Minute, wird zusätzlich das mechanische Bremssystem (`Bremse`) aktiviert. Dieses wird wieder deaktiviert, sobald die Umdrehungsgeschwindigkeit auf 16 Umdrehungen pro Minute oder weniger sinkt. In diesem Fall wird nicht direkt in den Normalzustand, sondern in den Zustand mit gedrehten Rotorblättern gewechselt, um abzuwarten, ob der Rotor wieder in den Überlastbereich beschleunigt.

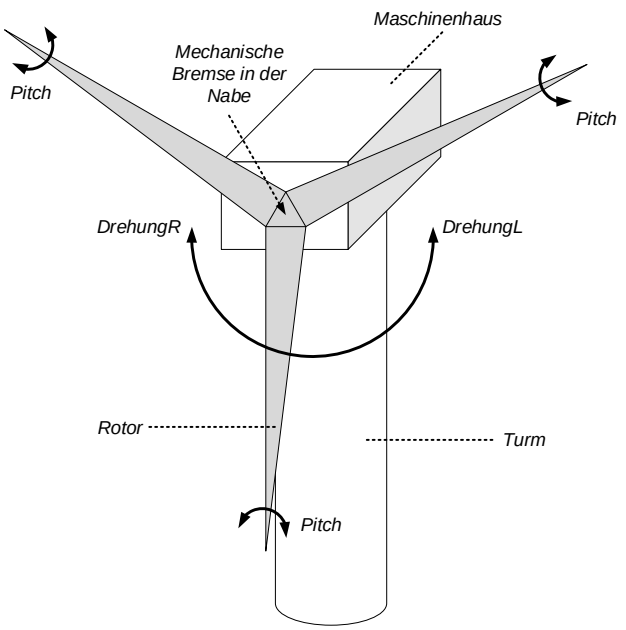


Bild 19.1: Skizze der Windkraftanlage

Vorgegeben ist das in Bild 19.2 gezeigte Zustandsdiagramm mit den vorgegebenen Zustandsnummern. Füllen Sie die durch römische Ziffern gekennzeichneten Lücken im Lösungsfeld aus.

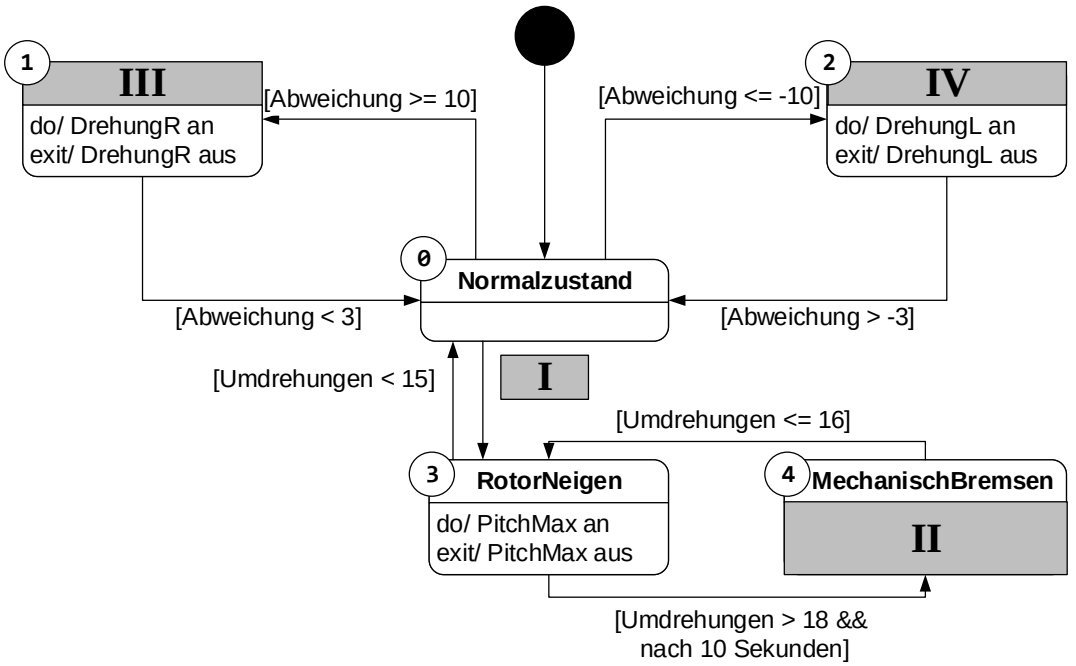


Bild 19.2: Zustandsdiagramm der Windkraftanlage

I. Geben Sie die korrekte Wächterbedingung an:
[Umdrehungen > 15 && nach 30 Sekunden]

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

MechanischBremsen

do/ Bremse ein

exit/ Bremse aus

III. Benennen Sie den Zustand korrekt: **DrehenRechts**

IV. Benennen Sie den Zustand korrekt: **DrehenLinks**

Warum ist im Zustandsdiagramm der Windkraftanlage (Bild 19.2) zwar ein Startzustand enthalten aber kein Endzustand? Begründen Sie Ihre Antwort knapp:

- **Startzustand bestimmt, was nach Start des Programms passiert (notwendig)**
- **Zyklische Ausführung, kontinuierliche Steuerung der Anlage, kein Ende**

20. Zustandsdiagramm zu C-Code

Aufgabe 20:
22 Punkte

Sie haben nun die Aufgabe, Teile des in Aufgabe 19 modellierten Zustandsdiagramms in Form von C-Code zu implementieren. Hierfür stehen Ihnen die in Tabelle 20.1 dargestellten Ein- und Ausgänge sowie erweiterte Variablen zur Verfügung. Weiterhin sind die in Tabelle 20.2 zusammengefassten Funktionen bereits implementiert und erlauben die Interaktion mit der Hardware der Windkraftanlage.

Tabelle 20.1: Sensor- und Aktorvariablen der Windkraftanlage, sowie erweiterte Variablen

Typ	Name	Beschreibung
AKTOREN	`a.Bremse`	Schaltet die mechanische Bremse ein (1) oder aus (0).
	`a.DrehungL`	Dreht das Maschinenhaus nach links (1) oder stoppt Bewegung nach links (0).
	`a.DrehungR`	Dreht das Maschinenhaus nach rechts (1) oder stoppt Bewegung nach rechts (0).
	`a.PitchMax`	Dreht die Rotorblätter “aus dem Wind” (1), um Auftrieb und damit Rotordrehzahl zu verringern, oder dreht die Rotorblätter in den Wind (0).
SENSOREN	`s.Abweichung`	Abweichung zwischen aktueller Windrichtung und Ausrichtung des Maschinenhauses in Grad (°). Positive Werte bedeuten, dass der Wind von rechts relativ zur Maschinenhausposition kommt, negative Werte links.
	`s.Umdrehungen`	Gibt die aktuelle Umdrehungsgeschwindigkeit der Rotorachse in Umdrehungen pro Minute (1/min) an.
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 20.2: Bereitgestellte Funktionen zur Interaktion mit der Hardware

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

a) Vervollständigen Sie das im folgenden Lösungskästchen gezeigte Programmgerüst, um eine zyklische Ausführung des Zustandsautomaten zu ermöglichen. Verwenden Sie hierfür die in Tabelle 20.1 angegebenen Variablennamen, da diese in anderen Programmteilen so verwendet werden sollen. Zur Interaktion mit der Hardware verwenden Sie die in Tabelle 20.2 gegebenen Funktionen, welche im Header *Windkraftanlage.h* bereits definiert und implementiert sind. Der Platzhalter `/* ZUSTAENDE */` soll später durch den spezifischen Code der Zustände in Form eines Zustandsautomaten ersetzt werden.

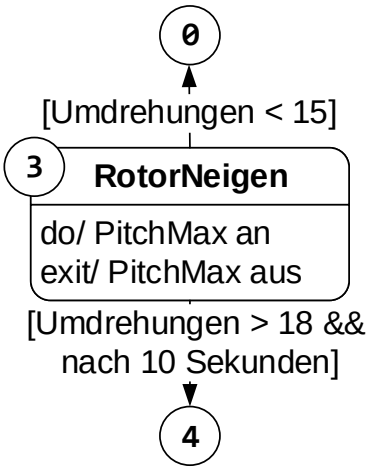
```
#include "Windkraftanlage.h„          //Headerimport

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

int main()
{
    unsigned int vplcZeit;  // Zeitvariable
    int schritt;            // Schrittvariable

for (;;) / while(1) / do // Zyklische Ausfuehrung
    {
        // Einlesen von Hardware
        leseEingaenge( &s, &vplcZeit   );
        switch (schritt)
        {
            /* ZUSTANDE */
        }
        // Schreiben auf Hardware
        schreibeAusgaenge( &a      );
    } while(1) falls oben do
    return 1; // Wird nicht erreicht
}
```

b) Der Zustand RotorNeigen (3) soll nun implementiert werden. Einige Codefragmente wurden bereits von einem Kollegen geschrieben. Vervollständigen Sie den vorliegenden Code im Lösungskästchen. Beachten Sie hierbei, dass mehr Leerzeilen zur Verfügung stehen, als für die korrekte Antwort erforderlich sind. Verwenden Sie die in Tabelle 20.1 angegebenen und in Aufgabe 20 a) implementierten Variablen.




```
case 3: //RotorNeigen
    a.PitchMax = 1;
    ____________________________
    ____________________________
    if (t == 0          )
    {
        t = vplcZeit + 10000; auch t = vplcZeit
        ____________________________________
        ____________________________________
        ____________________________________
    }
    else if (vplcZeit > t && s.Umdrehungen > 18
                Auch vplcZeit > t + 10000 … )
    {
        t = 0;
        ____________________________________
        a.PitchMax = 0;
        ____________________________________
        ____________________________________
        schritt = 4;
    }
    if (      s.Umdrehungen < 15
        ____________________________________)
    {
        t = 0;
        ____________________________________
        a.PitchMax = 0;
        ____________________________________
        ____________________________________
        schritt = 0;
    }
    break;
```

Aufgabe 21:
43 Punkte

21. Algorithmen und Datenstrukturen

Sie sollen nun schrittweise die Regelung für das zuvor behandelte mechanische Bremssystem auslegen.

a) Bitte ergänzen Sie zunächst die Komponenten eines mechatronischen Systems. Füllen Sie hierfür die in Bild 21.1 markierten Lücken (schwarze Linien) aus.

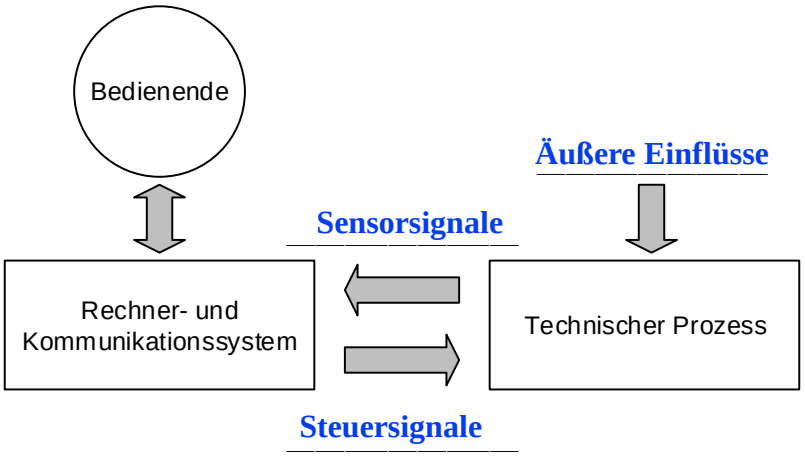


Bild 21.1: *Komponenten eines mechatronischen Systems*

b) Bild 21.2 zeigt die Einbindung von Sensordaten in ein mechatronisches System. Ein zentraler Bestandteil ist dabei die Wandlung der physischen Prozessgröße im Sensor und die Wandlung des analogen Sensorsignals in ein digitales Signal, das an den Bus angebunden werden kann.

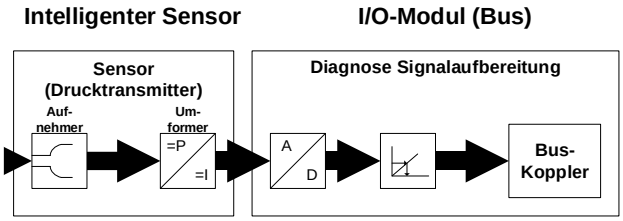


Bild 21.2: *Wandlung von Sensordaten zur Einbindung in ein mechatronisches System*

Nennen Sie jeweils ein Merkmal, welches die Genauigkeit der Umwandlung im Sensor beeinflusst, und ein Merkmal, dass die Genauigkeit der Digitalisierung beeinflusst. Begründen Sie den Einfluss und beschreiben Sie, wie die Genauigkeit erhöht werden kann.

Qualität des Sensors: **Genaueren Sensor verwenden**
Messprinzip: **Passendes Messprinzip wählen**

Wortbreite: **Erhöhung für geringeren Fehler**
Rauschverhalten A/D-Wandler: **Besseren Wandler**

c) Die Regelung des Bremssystems wurde im Programmcode (vgl. Zustandsdiagramm in Aufgabe 19) als *Zweipunktregler mit Hysterese* umgesetzt.

Bitte erläutern Sie zunächst den Vorteil eines *Zweipunktreglers mit Hysterese* gegenüber eines *Zweipunktreglers ohne Hysterese*.

Häufige Schaltvorgänge beim Zweipunktregler ohne Hysterese, Verwendung einer Hysterese verringert die Schaltvorgänge.

Sie wollen nun den Zweipunktregler mit Hysterese implementieren.

- **Beachten Sie, dass die Bremse nur zwei Zustände kennt (aus (0) und ein (1))**.
- Bedenken Sie beim Rückgabewert der Funktion `zpRegler` die Charakteristik des Steuersignals.
- Als Funktionsparameter soll `fAbweichung` dienen. Geben Sie einen passenden Datentypen an.

```
int  zpRegler(  float  fAbweichung)
{
    const int iHysterese = 50;

    if (fAbweichung > (iHysterese / 2))
        return 0;
    else if (fAbweichung < -(iHysterese / 2))
        return 1;
}
```

In einer anderen Variante des Windrades ist nun ein Generator verbaut, dessen Bremsleistung Sie stufenlos zwischen 0% und 100% regeln können. Bitte erläutern Sie kurz, warum in diesem Fall der Zweipunktregler nicht optimal ist und welche Regler Sie stattdessen einsetzen möchten.

Zweipunktregler ist nur geeignet für ein diskretes Ausgangssignal, bei kontinuierlicher Regelung müssen entsprechende Regler verwendet werden (z.B. I-Regler, PI-Regler).

d) Sie möchten die Steuerung nun mit einem P-Regler implementieren. Geben Sie die Datentypen für die beiden Variablen an. Berechnen Sie `fStellgroesse` nach dem Prinzip eines P-Reglers.

```
// ...
float___________ pRegler(float fAbweichung)
{
    float________ fVerstaerkung = 2.7;
    float________ fStellgroesse = 1.0 ;
    fStellgroesse = _fVerstaerkung * fAbweichung_____;
    return fStellgroesse;
}

// ...
```

e) Sie möchten nun den Regler in ein zyklisch ablaufendes Programm einbinden. In jedem Zyklus lesen Sie zunächst die Sensoren ein, rufen die Reglerfunktion (`pRegler()`) auf, um Ihre Stellgröße zu ermitteln, und setzen diese dann. Zusätzlich möchten Sie in jedem Zyklus die Stellgröße in eine log-Datei mit dem Dateinamen `log.txt` speichern. Sollte die Datei bereits vorhanden sein, soll sie überschrieben, andernfalls neu erstellt werden.

Füllen Sie die Lücken im folgenden Code. Um das Beenden der Schleifen müssen Sie sich nicht kümmern. Weiterhin stehen Ihnen folgende Funktionen und Variablen zur Verfügung, die Sie bei Ihrer Implementierung verwenden sollten.

```
float drehzahlLesen();// Liest aktuelle Drehzahl

float pRegler(float Abweichung); // Ruft P-Regler auf und gibt
                                 Stellwert zurück
```

Folgende Variablen sind bereits definiert, zugewiesen und können von Ihnen verwendet werden:

```
Float fSolldrehzahl // Sollwert der Drehzahl

Float fDrehzahl     // Aktuell gemessene Drehzahl
```

```
//
FILE*_______ logFile = __fopen___("log.txt"__, __"w"__);
// Logdatei oeffnen
while(1)
{
  fDrehzahl = ____drehzahlLesen()_____________________;
  fStellgroesse = pRegler(fSolldrehzahl - fDrehzahl);

  fprintf(logFile, "%f", fStellgroesse)_____;
  // Schreiben in log-Datei
  // Weiterer Code
}

fclose(logFile)_____; // Log schliessen
```

f) Der P-Regler funktioniert für Ihre Anwendung. Allerdings führt er zu einer bleibenden Regelabweichung, die in Ihrem Anwendungsfall nicht akzeptabel ist. Erklären Sie im Folgenden die Grundidee des PI-Reglers und wie dieser die Regelabweichung minimieren kann.

Grundidee:

Einbeziehen der Regelabweichung in der Vergangenheit durch Integral.
Dadurch kann die Abweichung mit der Zeit auf 0 reduziert werden.

Sie wollen nun auch eine Funktion zum Berechnen der Integrale schreiben. Die Schnittstelle der Funktion ist bereits vorgegeben. Die Funktion übernimmt einen Pointer auf den letzten Funktionswert in einem Array. Dieses Array umfasst fünf Elemente (Fensterbreite 5) und beinhaltet die letzten fünf Regelabweichungen $e(t)$ mit dem aktuellen Wert an letzter Position. Die Schrittweite zwischen den Punkten beträgt 1s (Variable `iDelta`):

$$e(t_0 - 4\,\mathrm{s});\ e(t_0 - 3\,\mathrm{s}); e(t_0 - 2\,\mathrm{s}); e(t_0 - 1\,\mathrm{s});\ e(t_0)$$

Füllen Sie die Lücken und implementieren Sie die Funktion für die Annäherung des Integrals. **Anzahl und Länge der Lücken lassen keinen Rückschluss auf die korrekte Lösung zu.**

```
float integrieren(float* valP)
{
  const int iDelta = 1; // In Sekunden
  float value = 0.0;    // Berechnetes Integral

  _____________________________________________
  for (int i = 4; i >= 0; i--) {
  _____________________________________________
      value +=  *(valP - i) * iDelta;
  _____________________________________________
  }
  _____________________________________________
  _____________________________________________
  _____________________________________________
  _____________________________________________
  _____________________________________________
  return value;
}
```