Prüfung
„Grundlagen der modernen Informationstechnik“
Wintersemester 2021/2022

11.03.2022

Musterlösung
- ohne Gewähr -

Aufgabe 1: Grundlagen der Informationstechnik und Digitaltechnik

a) Gegeben ist das dargestellte kontinuierliche Signal f(t). Die Diskretisierung des kontinuierlichen Signals ist für die Zeitpunkte t_0, t_1 gegeben und als (📍) dargestellt. Identifizieren und kreuzen Sie die Art der Diskretisierung und die Art der Rundung der Funktionswerte f(t) an. Setzen Sie die Diskretisierung für die Zeitpunkte $t_i, i \in$ [2, ..., 7] im Koordinatensystem nach denselben Regeln wie bei t_0, t_1 fort, indem Sie die Werte in der Grafik markieren.

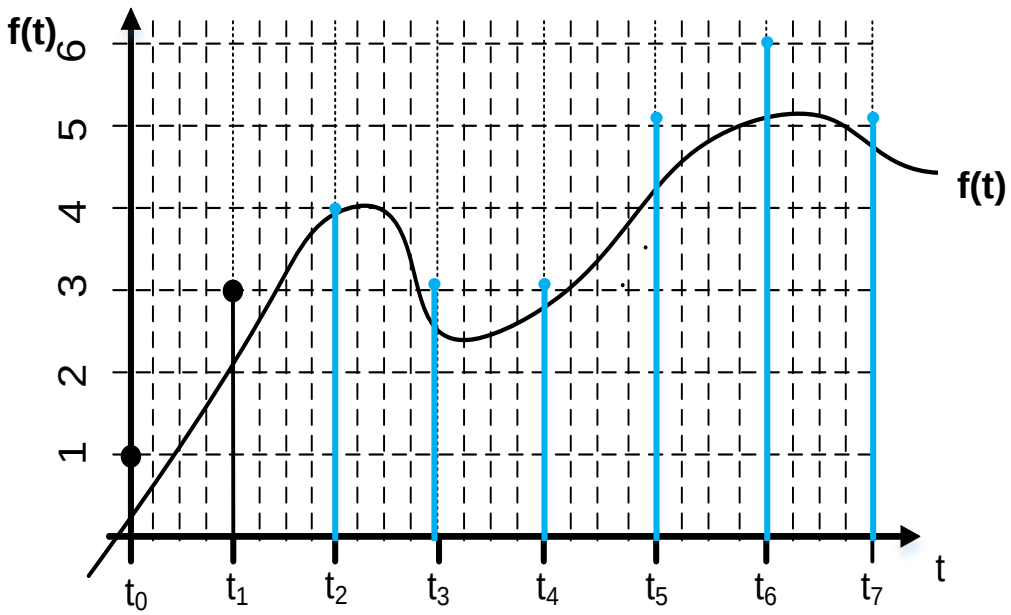
1. Um welche Art der Diskretisierung handelt es sich?

- ☐ wertkontinuierlich und zeitdiskret
- ☒ wert- und zeitdiskret
- ☐ wertkontinuierlich und zeitkontinuierlich
- ☐ wertdiskret und zeitkontinuierlich

2. Art der Rundung: Kreuzen Sie exemplarisch an, auf welchen diskreten Wert der Funktionswert f(t) = 2.3 gerundet wird.

- ☐ 2.5
- ☐ 2
- ☐ 2.3
- ☒ 3

3. Vervollständigen Sie die Diskretisierung von f(t):



b) Zahlensysteme: Lösen Sie die nachfolgend dargestellte Rechnung und geben Sie die Lösung im Binär- sowie im Dezimalsystem an.

$$(121)_9 + (15)_7 = (\ 1110000\ )_2 = (\ 112\ )_{10}$$

Aufgabe 2: IEEE 754 Gleitkommadarstellung

a) Rechnen Sie die gegebene Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) in eine Dezimalzahl um, indem Sie die folgenden Textblöcke ausfüllen.

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

1	1	0	1	0	0	1	1	0	1	0	0
V	E (4 Bit)				M (7 Bit)						

Vorzeichen

1 = “negativ”

Bias als Dezimalzahl

$B = 2^{(x-1)} - 1 = 7$

Biased Exponent als Dezimalzahl

$E = (1010)_2 = (10)_{10}$

Exponent als Dezimalzahl

e = E – B = 3

Vollständige Dualzahl (denormalisierte Mantisse) ohne Vorzeichen

$= (101101)_2 * 2^3 = (1011{,}01)_2$

Vollständige Dezimalzahl (inkl. Vorzeichen)

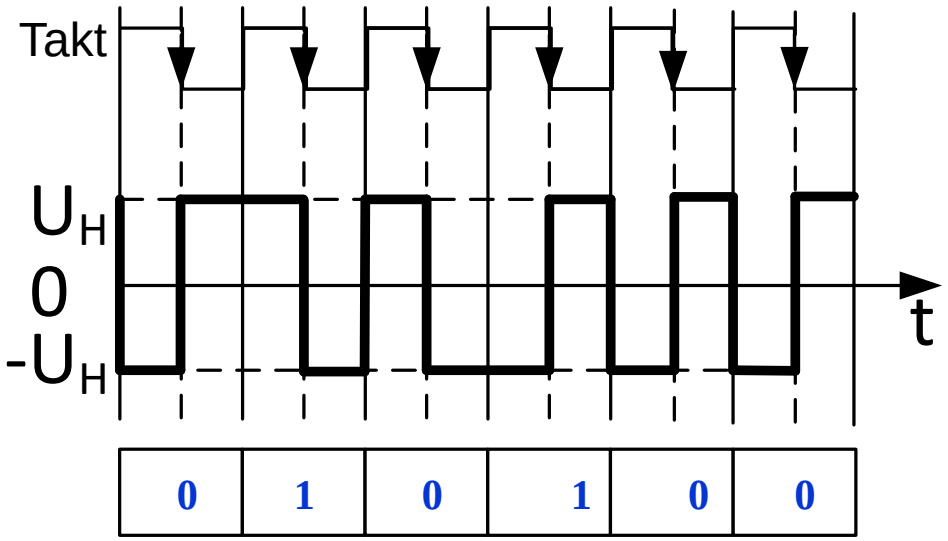
-11,25

b) Wie lautet die positive Gleitkommadarstellung (angelehnt an die IEEE 754 Darstellung) der Zahl $(1110000)_2 = (112)_{10}$?

0	1	1	0	1	1	1	0	0	0	0	0
V	E (4 Bit)				M (7 Bit)						

Aufgabe 3: Differential-Manchester-Code und Hamming Distanz

a) Das nachfolgend dargestellte Signal wurde mit Hilfe des Differential-Manchester-Codes für eine Übertragung mittels seriellem Bussystem erzeugt. Welche Binärfolge liegt dem dargestellten Signal zu Grunde?



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

b) Die Hamming-Distanz ist ein Maß für die Störsicherheit eines Codes. Bestimmen Sie die Hamming-Distanz h anhand der zwei abgebildeten Codewörter, die Anzahl e an Bitfehlern, welche erkannt werden können und die Anzahl f an Bitfehlern, welche behoben werden können.

Codewort 1	1	1	0	0	1
Codewort 2	1	0	0	1	0
	-	X	-	X	X

h = 3

e = 3 – 1 = 2

f = (3 – 1) / 2 = 1

Aufgabe 4: Logische Schaltungen und Schaltbilder

a) Gegeben ist das Schaltbild der vollständigen disjunktiven Normalform (DNF). Vervollständigen Sie die zugehörige, nebenstehende Wahrheitstabelle (Tabelle 4.1).

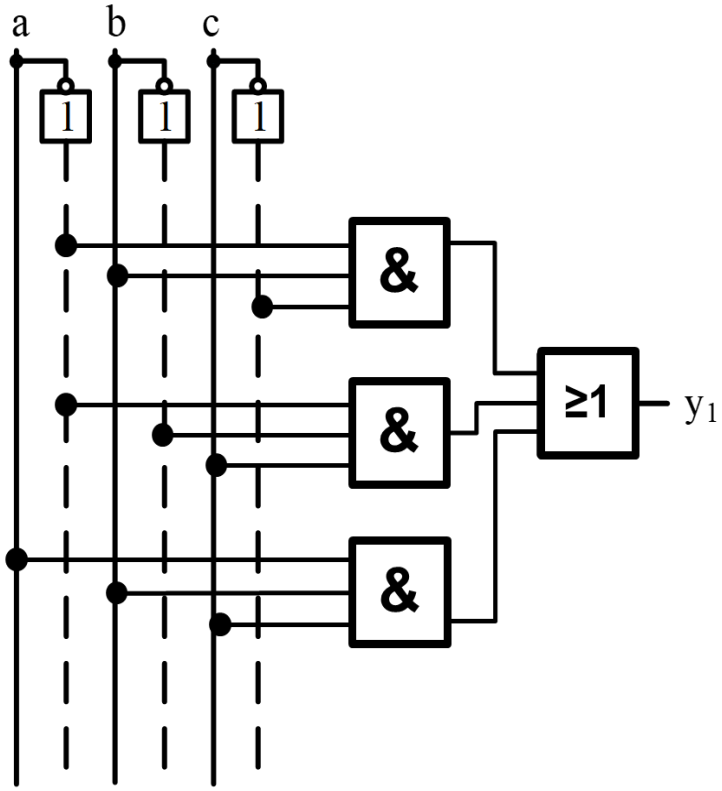


Tabelle 4.1: *Wahrheitstabelle:*

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

b) Welche Schaltung ist hier dargestellt? Bitte kreuzen Sie den richtigen Fachbegriff

( ) Dreikanal Demultiplexer

( ) OR-Gatter für 3 Eingänge

( ) Zweikanal-Multiplexer mit Selektionseingang „b“

( ) XOR-Gatter für 3 Eingänge

(**X**) Keine der Antwortmöglichkeiten ist für die obige Schaltung zutreffend.

Aufgabe 5: FlipFlops

Gegeben sind die zeitlichen Verläufe (Bild 6.1) der Eingangssignale A, B, und T sowie des Ausgangssignals Q_1 einer Master-Slave-Flipflop-Schaltung (Bild 6.2).

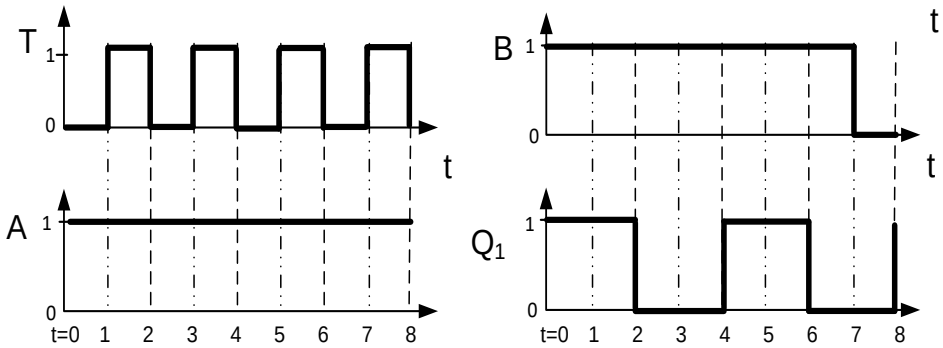


Bild 5.1: *Zeitliche Verläufe des MS-FF*

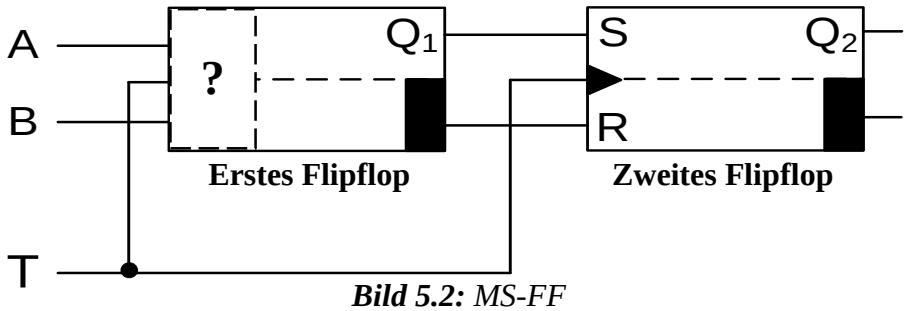


Bild 5.2: *MS-FF*

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

a) Bestimmen Sie mithilfe der in Bild 5.1 gegebenen Verläufe die Art des ersten Flipflops des Schaltbilds (Bild 5.2) sowie dessen zugehörige Art der Taktsteuerung.

b) Tragen Sie den zeitlichen Verlauf für Q_2 in das vorgegebene Koordinatensystem im Lösungsfeld ein.

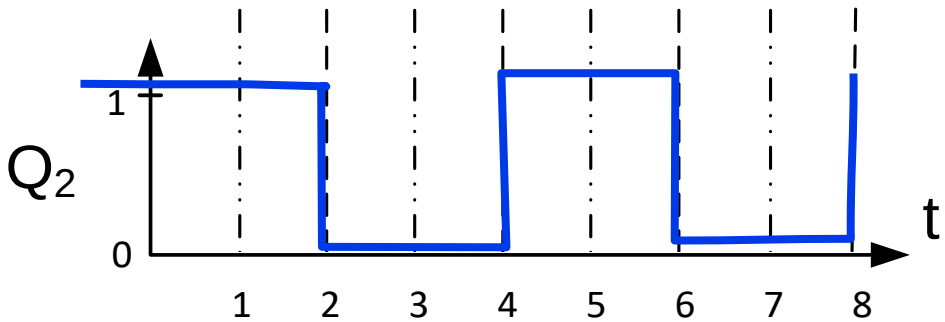
Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden.

a) Art und Taktsteuerung des ersten Flipflops (markiert durch Fragezeichen in Bild 5.2):

J K **- Flipflop**

Negative **Takt-** **flanken** **-steuerung**

b) Verlauf:

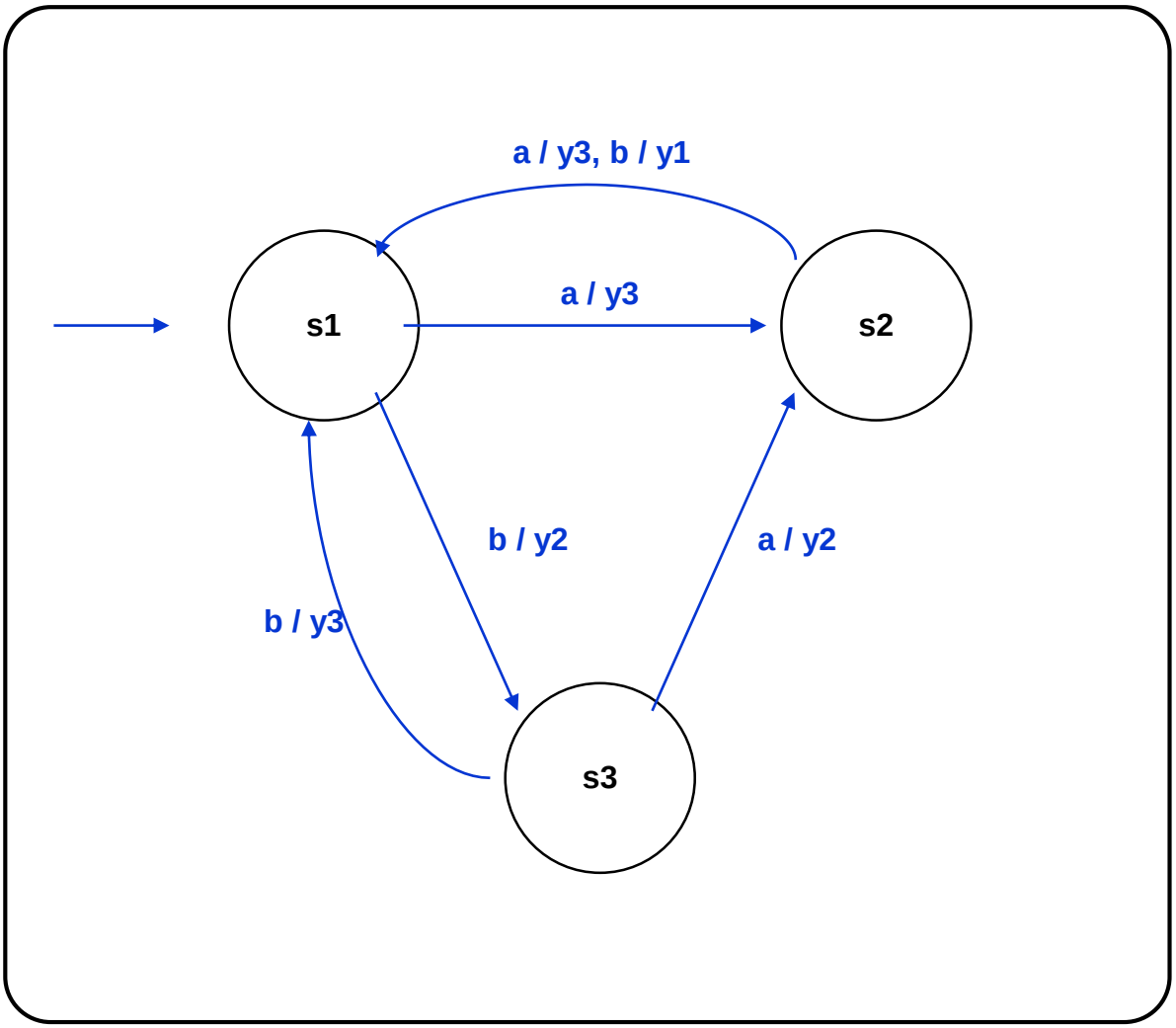


Aufgabe 6: Automaten

Gegeben sei die folgende Übergangstabelle mit den Zuständen s1, s2 und s3, den möglichen Eingaben a und b und den Ausgaben y1, y2, y3.

T \ S	s1	s2	s3
a	s2, y3	s1, y3	s2, y2
b	s3, y2	s1, y1	s1, y3

Vervollständigen Sie die untenstehende Grafik unter der Annahme, dass s1 der Startzustand ist.



Aufgabe 7: MMIX – Teil 1

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: $\frac{b-(256*a)^2}{c}+32$

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 68 72	Zwischenergebnis
$0x8B	0x00 00 00 00 00 00 01 00	Nicht veränderbar
…	…	

Bild 7.2: *Registerspeicher*

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	MUL $0x8A, $0x87 $0x8B
2	MUL $0x8A, $0x8A, $0x8A
3	SUB $0x8A, $0x88, $0x8A
4	DIV $0x8A, $0x8A, $0x89
5	ADDI $0x8A, $0x8A, 0x20

Aufgabe 8: MMIX – Teil 2

a) Nehmen Sie an, der Inhalt des Datenspeichers entspricht dem Zustand in Bild 7.3. Laden Sie ein Tetra ab Speicherstelle 0x0 … 63 0D in die Variable c im Registerspeicher (Big Endian).

Geben Sie genau einen, hierfür notwendige Befehl in Assembler-Code an. Wie lautet der 64-Bit-Wert der Variablen c nach dem Ladevorgang?

Befehl:
LDT $0x89 $0x86 $0x8B

Alt.:
LDTI $0x89 $0x86 0xFF
LDTI $0x89 $0x86 0xFE
LDTI $0x89 $0x86 0xFD

Wert (64 Bit):

0x 00 00 00 00 BA 98 76 54

Datenspeicher	
Adresse	Wert
…	…
0x00 00 00 00 00 00 63 07	0x54
0x00 00 00 00 00 00 63 08	0x32
0x00 00 00 00 00 00 63 09	0x10
0x00 00 00 00 00 00 63 0A	0xFE
0x00 00 00 00 00 00 63 0B	0xDC
0x00 00 00 00 00 00 63 0C	0xBA
0x00 00 00 00 00 00 63 0D	0x98
0x00 00 00 00 00 00 63 0E	0x76
0x00 00 00 00 00 00 63 0F	0x54
0x00 00 00 00 00 00 63 10	0x32
0x00 00 00 00 00 00 63 11	0x10
…	…

Bild 7.3: *Datenspeicher*

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

INCMH $0x01 0x0203

c) **Bussysteme und Prozessperipherie**

Kreuzen Sie die jeweils richtige Antwort an.

Bei welchen Netzwerktopologien…

…sind alle Knoten über genau eine gemeinsame Leitung verbunden?

Stern ( ) Baum ( ) Keine der Antworten ist richtig (**X**)

… besitzt genau ein einziger Knoten eine direkte Verbindung zu allen anderen Knoten?

Bus ( ) Linie ( ) Stern (**X**)

Aufgabe 9: Echtzeitprogrammiersprache PEARL

Ein Fahrrad hat einen Task „Fahren“ mit einer Unterbrechung „Druckverlust“. In dem Task „Fahren“ wird der Task „Messen“ alle 10 Sekunden aktiviert. Der Task „Ausgeben“ wird aktiviert. Erkennt der Sensor im Rad einen Druckverlust (Unterbrechung), so wird der Task „Stoppen“ aktiviert, welcher „Messen“ beendet und ausplant sowie „Ausgeben“ unterbricht (RTOS-UH-Taktzustandsdiagramm).

Vervollständigen Sie den untenstehenden Codeausschnitt in PEARL.

```
// Definiere Task Stoppen mit Priorität 9
Stoppen : TASK PRIORITY 9 ;

  // Zustaende für Messung und Ausgeben nach RTOS-UH
  TERMINATE Messen;
  PREVENT Messen;
  SUSPEND Ausgeben;
 END;

// Definiere Task Fahren mit Priorität 12
Fahren : TASK PRIORITY 12 ;

  // Spezifiziere Unterbrechung „Druckverlust“
  SPC Druckverlust INTERRUPT ;

  // Aktiviere Unterbrechung „Druckverlust“

  ENABLE Druckverlust;

  // Aktiviere Task Messen alle 10 Sekunden

  ALL 10 SEC ACTIVATE Messen;

  ACTIVATE Ausgeben;


  WHEN Druckverlust ACTIVATE Stoppen;

  // Ende des Tasks
  END;
```

Aufgabe 10: Scheduling

Gegeben seien die folgenden fünf Prozesse (A-E, Tabelle 9.1), welche jeweils ab dem Zeitpunkt *Start* (gemessen ab t=0) eingeplant werden sollen. Zur Abarbeitung eines Tasks wird die Rechenzeitspanne *Dauer* benötigt. Der Ist-Verlauf (Bild 9.1) zeigt das Scheduling nach dem Schema Round-Robin für den Zeitraum $t = [0; 18[$ s für einen Einkernprozessor.

Beantworten Sie die untenstehenden Fragen. Geben Sie, sofern gefragt, die Reihenfolge der ablaufenden Tasks an, z.B. ABCB.

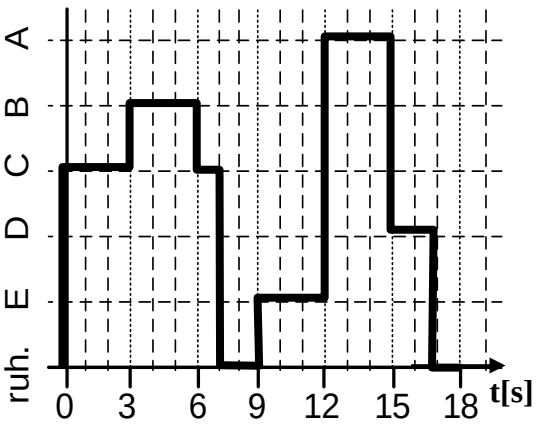


Bild 9.1: *Ist-Verlauf der Tasks*

	Priorität	Start-zeitpunkt	Dauer (ab Start)	Zeitpunkt Deadline
A	1 (hoch)	5 s	3 s	11 s
B	2	1 s	11 s	17s
C	3	0 s	4 s	5s
D	4	5 s	2 s	11s
E	5 (niedrig)	4 s	3 s	7s

Tabelle 9.1: *Taskspezifikation*

Art des Schedulings (präemptiv / nicht-präemptiv):

präemptiv

Größe des Zeitschlitzes in Sekunden:

3 sek

Scheduling Strategien für das Eintreffen mehrere Tasks innerhalb eines Zeitschlitzes:

zuerst FIFO und anschließend die Prioritäten

Geben Sie die Reihenfolge der Tasks für das nicht-präemptive Scheduling Verfahren FIFO an. Tasks mit gleichem Startzeitpunkt werden nach Prioritäten geschedult.

CBEAD

Wird bei einem nicht-präemptiven Earliest Deadline First (EDF) Scheduling die Bedingung der Rechtzeitigkeit aller Tasks erfüllt? Begründen Sie ihre Antwort und geben Sie die Reihenfolge der Tasks an (bei gleicher Deadline erfolgt Scheduling nach Prioritäten):

CEADB, Nein, da B und D die Deadline nicht halten.

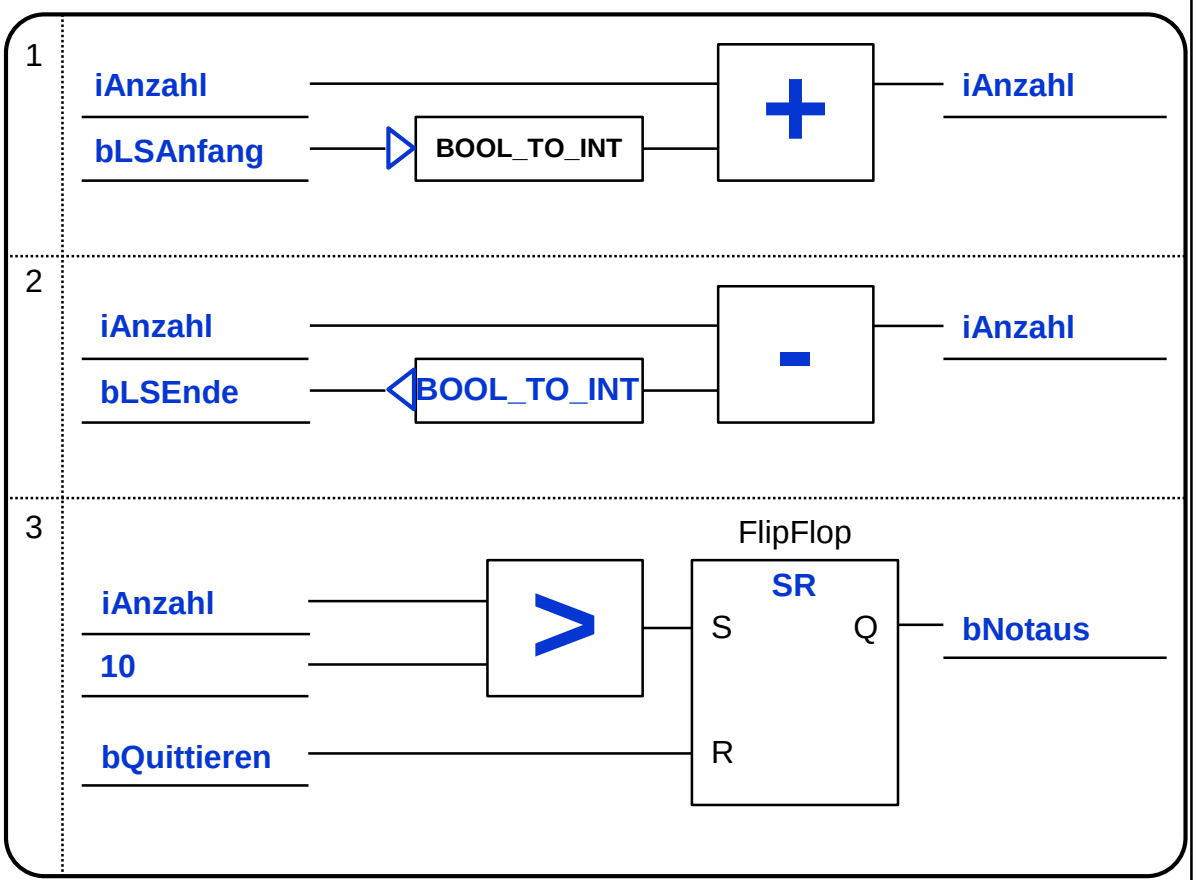
Aufgabe 11: IEC FBS

Auf einem Förderband fahren Werkstücke in den Arbeitsraum einer Maschine. Der Arbeitsraum ist an Beginn und Ende durch Sensoren überwacht (*bLSAnfang*, *bLSEnde*).

- Beginnt ein Werkstück den Eingang zum Arbeitsraum zu passieren, wird der Zähler *iAnzahl* um 1 erhöht.
- Hat ein Werkstück den Arbeitsraum **vollständig** verlassen, wird der Zähler *iAnzahl* um 1 verringert.
- Ist die Anzahl Werkstücke im Arbeitsraum *iAnzahl* größer als *10*, wird *bNotaus* auf TRUE gesetzt und muss zum Zurücksetzen explizit quittiert werden (*bQuittieren*).
- Um *bNotaus* zurückzusetzen, muss der Arbeitsraum geleert und *iAnzahl* zurückgesetzt werden (*iAnzahl <= 10*), bevor *bQuittieren* entgegengenommen wird.

Hinweis:
- *BOOL_TO_INT gibt 1 aus, wenn am Eingang TRUE anliegt. Sonst ist die Ausgabe 0.* ***In FBS werden Datentypen nicht automatisch konvertiert (wie in C/C++).***
- *Signalverzögerungen im System sind zu vernachlässigen.*
- *Verwenden Sie* ***keine*** *Schaltglieder außer den in der Vorlage bereits vorhandenen.*
- *Ergänzen Sie Negationen und Flankenerkennung falls notwendig.*
- *Statt ≥ bzw. ≤ können Sie auch >= bzw. <= schreiben.*

Ergänzen Sie die untenstehenden Programme so, dass das oben beschriebene Verhalten erfüllt wird.



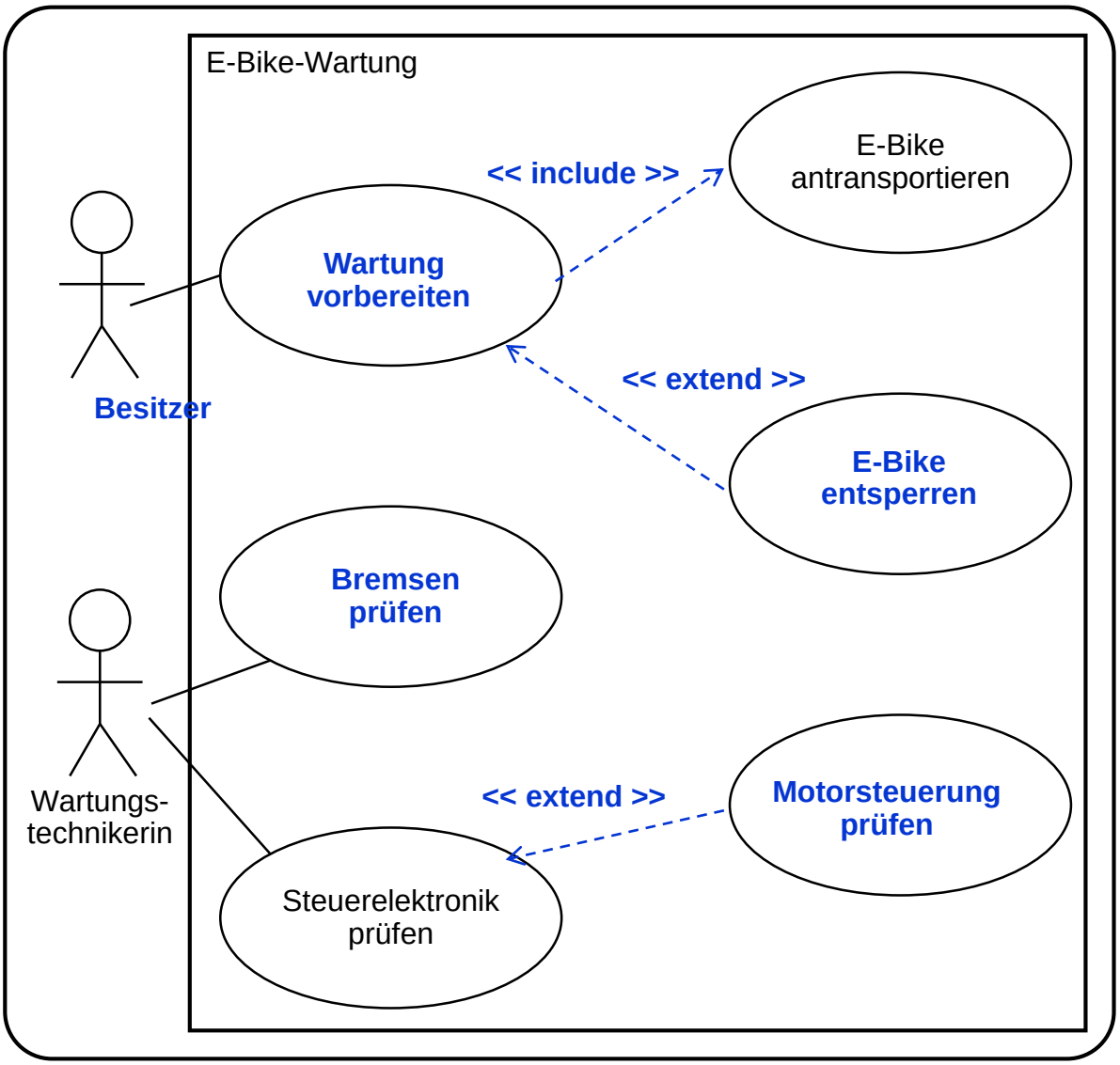
Aufgabe 12: Use-Case-Diagramm

Moderne E-Bikes verfügen über eine große Anzahl interagierender technischer Komponenten. Um zu garantieren, dass trotz der Komplexität ein sicheres Fahren mit dem E-Bike möglich ist, sollen Sie es in den folgenden Aufgaben mithilfe der UML modellieren und seine Software in C++ implementieren.

Ein E-Bike wird von seinem *Besitzer* zur Wartung gebracht. Dazu muss er die *Wartung vorbereiten*, was die Hinfahrt (*E-Bike antransportieren*) einschließt. Falls die Steuerung des E-Bikes durch eine Pin gesichert wurde, ist zusätzlich *E-Bike entsperren* notwendig.

Die beauftragte *Wartungstechnikerin* muss nun unter anderem die *Bremsen prüfen* und die *Steuerelektronik prüfen*. Abhängig davon, ob bei der Prüfung der Steuerelektronik Probleme auftreten, ist zusätzlich eine detailliertere Prüfung der Motorsteuerung nötig (*Motorsteuerung prüfen*).

Füllen Sie mithilfe der obigen Angaben das Use-Case-Diagramm der UML für die Wartung des Fahrrads aus. Benennen Sie die Akteure sowie die Anwendungsfälle. Bitte ergänzen Sie auch die Verbindungen zwischen den Use-Cases mit richtiger Beziehungsart und Richtung (Pfeilspitze).

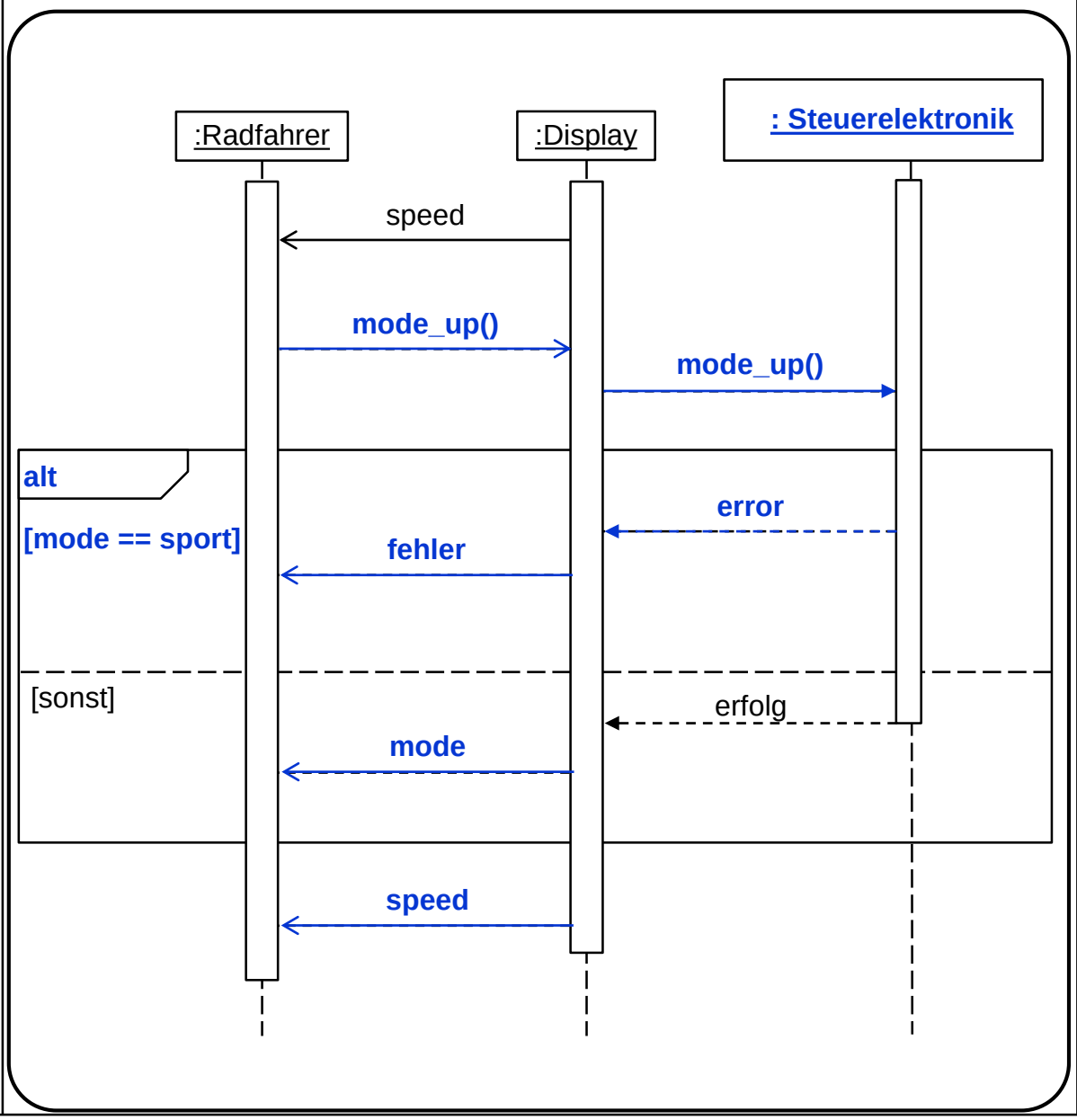


Aufgabe 13: Sequenzdiagramm

Das *Display* kommuniziert mit dem *Radfahrer* ausschließlich über asynchrone Nachrichten. Es zeigt ihm die aktuelle Geschwindigkeit (Wert *speed*) an. Der *Radfahrer* schaltet mit der Methode *mode_up()* in einen höheren Geschwindigkeitsmodus. Das Display gibt den Methodenaufruf *mode_up()* weiter an die *Steuerelektronik* und wartet auf deren Antwort. Falls das E-Bike bereits im Sportmodus ist (*mode* ist *sport*), wird die Antwort *error* von der Steuerelektronik ans Display gegeben. Das Display meldet dann *fehler* an den Radfahrer. Andernfalls führt die Antwort *erfolg* von Steuerelektronik an Display zur Ausgabe des neuen Modus (Nachricht Wert *mode*) an den Radfahrer. In jedem Fall gibt das Display anschließend wieder den aktuellen Geschwindigkeitswert (*speed*) an den Radfahrer. Nachrichten zwischen Display und Radfahrer blockieren deren sonstiges Verhalten nicht, sind also asynchron.

Ergänzen Sie das Sequenzdiagramm entsprechend der Beschreibung.

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



Aufgabe 14: Grundlagen in C

a) Deklarieren Sie ein Array namens `Speicher` mit drei Speicherplätzen vom Datentyp Integer. Initialisieren Sie das Array mit den Zahlen `25, 42` und `73` in aufsteigender Reihenfolge. Erhöhen Sie die letzte Speicherzelle um 5, unabhängig von Ihrem aktuellen Wert. Geben Sie den Wert der vorletzten Speicherzelle auf der Kommandozeile als dezimale Ganzzahl aus. Verzichten Sie auf die Verwendung von Pointer-Arithmetik.

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

```
int Speicher[3] = {25, 42, 73};
Speicher[2] = Speicher[2] + 5;
printf("%d", Speicher[1]);
alternative: printf("%i", Speicher[1]);
```

b) Gegeben Sei folgender Programmausschnitt. Kreuzen Sie an, ob die gegebenen Aussagen wahr (w) oder falsch (f) sind.

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

```
int i=5;
while(1)
{
        i = i + 1;
        if(i%5){
                printf("%i\n",i);
                continue;
        }
        /*else // else-Path
        {
                printf("%d\n",i);
        }
        */
}
```

	(w)	(f)
Die Variable i wird nie kleiner als 5.	☐	☒
Der Wert i kann auch negativ Werte annehmen, da ein Overflow stattfinden kann.	☒	☐
Wenn die if-Bedingung wahr ist, wird die Zahl i als Ganzzahl ausgegeben.	☒	☐
Wenn die if-Bedingung wahr ist, wird die while-Schleife verlassen.	☐	☒
Wenn die if-Bedingung falsch ist, wird die Zahl i als Ganzzahl ausgegeben.	☐	☒

c) Welchen Wert müssen B und C annehmen, damit folgender Ausdruck gilt:

```
(14 & B) >> C = 3
```

Kreuzen Sie an, ob die gegebenen Aussagen wahr (w) oder falsch (f) sind.

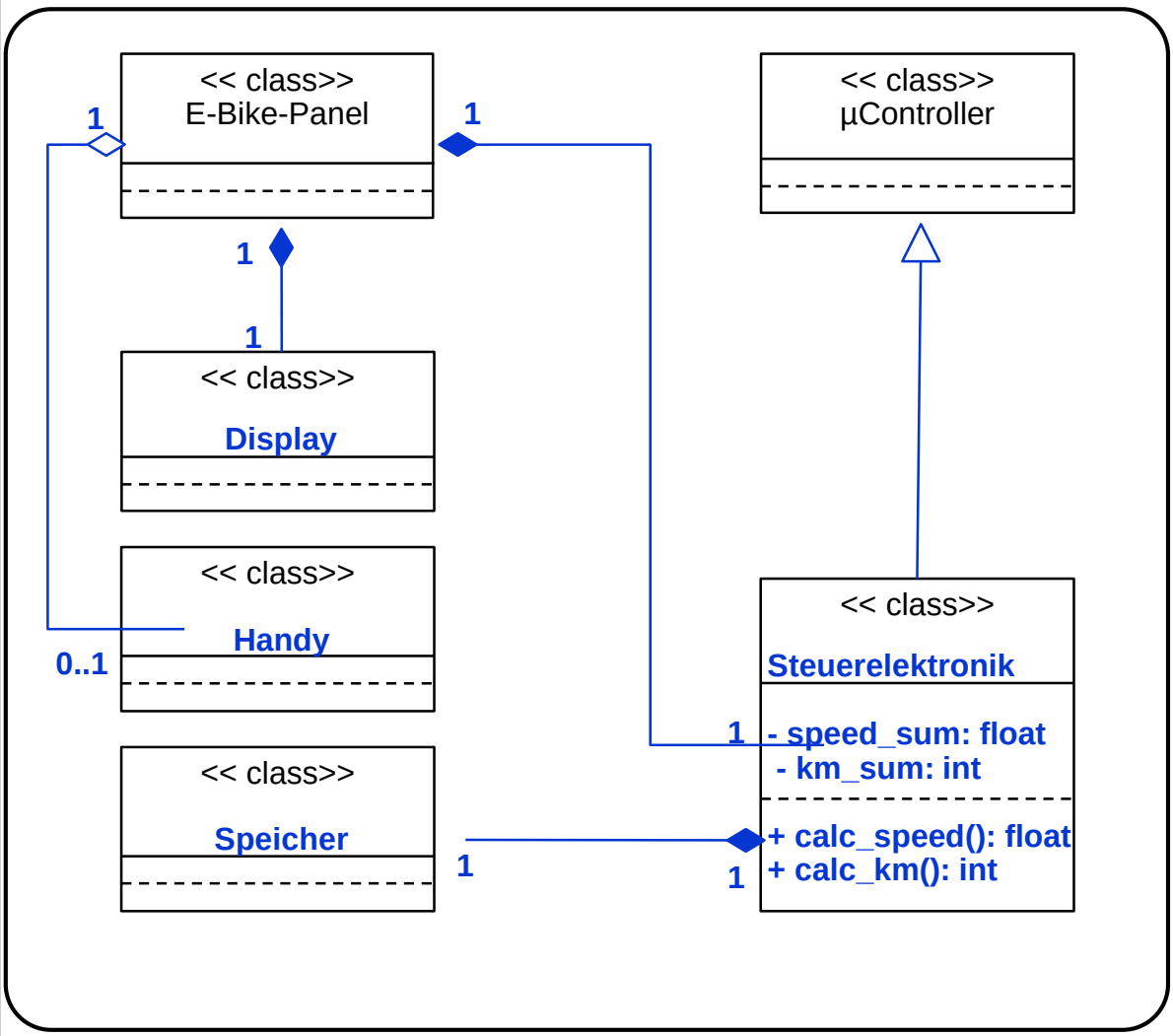
	(w)	(f)
B = 6; C = 1;	☒	☐
B = 8; C = 3;	☐	☒
B = 2; C = 1;	☐	☒

Aufgabe 15: Klassendiagramm (Objektorientierung)

Die Bedieneinheit eines E-Bikes (*E-Bike-Panel*) besteht aus einer *Steuerelektronik* und einem *Display*. **Optional** kann über ein Dock ein *Handy* angeschlossen werden.

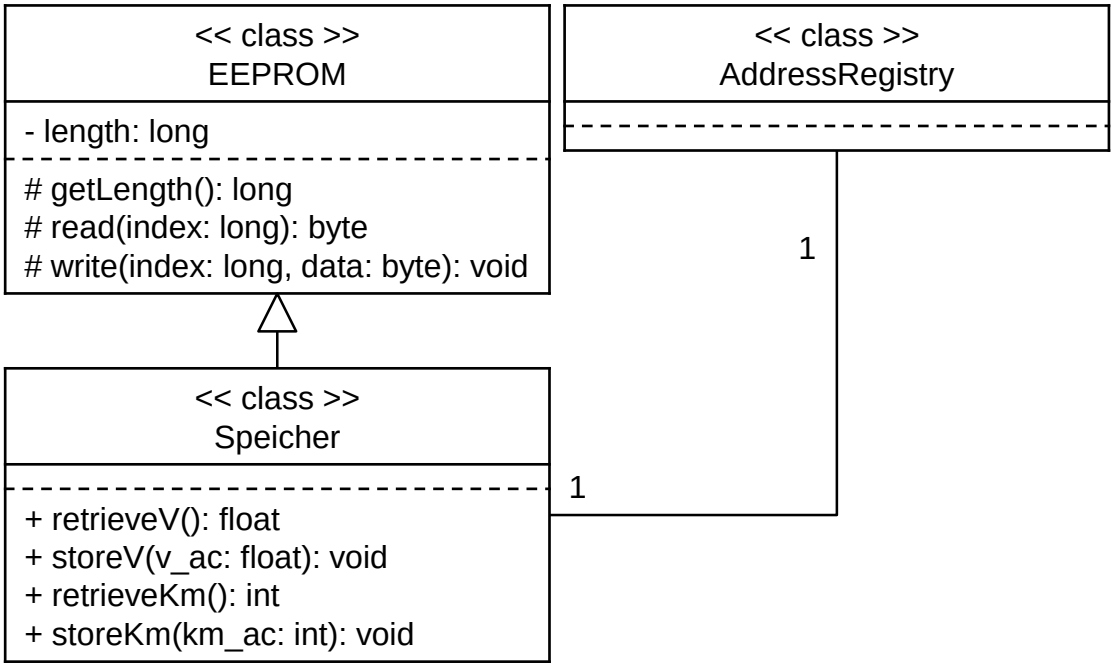
- Die Steuerelektronik ist eine Spezialisierung der Klasse *µController*.
- Die Steuerelektronik besteht unter anderem aus einem *Speicher*.
- Die Steuerelektronik bietet zwei öffentliche Operationen: *calc_speed()* gibt die aktuelle Geschwindigkeit als Gleitkommazahl zurück, *calc_km()* liefert die gefahrenen Kilometer als Ganzzahl.
- calc_speed() und calc_km() speichern die Ergebnisse ihrer Berechnung in den privaten Attributen *speed_sum* bzw. *km_sum* der Klasse Steuerelektronik, jeweils vom gleichen Datentyp wie die zugehörige Operation.

Vervollständigen Sie das untenstehende UML-Klassendiagramm. Die Angabe von Kardinalitäten ist nicht zu vernachlässigen.



Aufgabe 16: Klassendiagramm zu Code

Es wird nun der Speicher der Steuerelektronik näher betrachtet. Als Vorgabe dient Ihnen ein UML Klassendiagramm, das Sie in C++ übersetzen sollen.




```
class EEPROM {
  private:
    long length;
  protected:
    long getLength();
    byte read(long index);
  void write(long index, byte data);
};

class AddressRegistry {};

class Speicher : public EEPROM {
  private:
   AdressRegistry *addresses;
  public:
    float retrieveV();
    void storeV(float v_ac);
    int retrieveKm();
    void storeKm(int km_ac);
};
```

Aufgabe 17: Zustandsdiagramm

Im Folgenden wird die Betriebssoftware eines E-Bikes betrachtet, siehe **Abb. 17.1**. Bei einem E-Bike handelt es sich um ein Fahrrad, in dem zur Tritt-Unterstützung ein Elektromotor verbaut wird. Beim Start (Zustand `System initialisiert`) wird das System durch den **einmaligen** Aufruf der Methode `initsystem` initialisiert. Danach werden zwei am E-Bike fest verbaute Drehzahlmesser abgefragt (Zustand `messend`): Dazu wird die Geschwindigkeit `v` an der Gabel bestimmt (Methode `Vsen`) sowie die Trittfrequenz `ft` des Fahrenden (Methode `FTsen`) gemessen. Liefert einer der Sensoren falsche oder keine Werte (Wächterbedingung `Messung_fehlerhaft`), wird der Motor abgeschaltet (Zustand `Motor gestoppt`). Bei erfolgreicher Messung wird in Zustand `ft auswertend` übergegangen. Die Sensorwerte werden nacheinander in zwei Zuständen ausgewertet (Zustand `ft auswertend`, Zustand `v auswertend`):

- Eine Unterstützung durch den Motor ist nur zulässig, wenn der Fahrende selber mit tritt (`ft !=0`). Andernfalls ist der Motor abzuschalten (Zustand `Motor gestoppt`).
- Eine Unterstützung durch den Motor ist nur bis (inklusive) 25 km/h zulässig. Andernfalls wird der Motor abgeschaltet (Zustand `Motor gestoppt`).
- Ist eine Unterstützung durch den Motor zulässig, wird die Geschwindigkeit `v` einem PID-Regler (Methode `PIDregler`) übergeben (Zustand `Motor geregelt`). Die Methode `PIDregler` wird kontinuerlich aufgerufen.

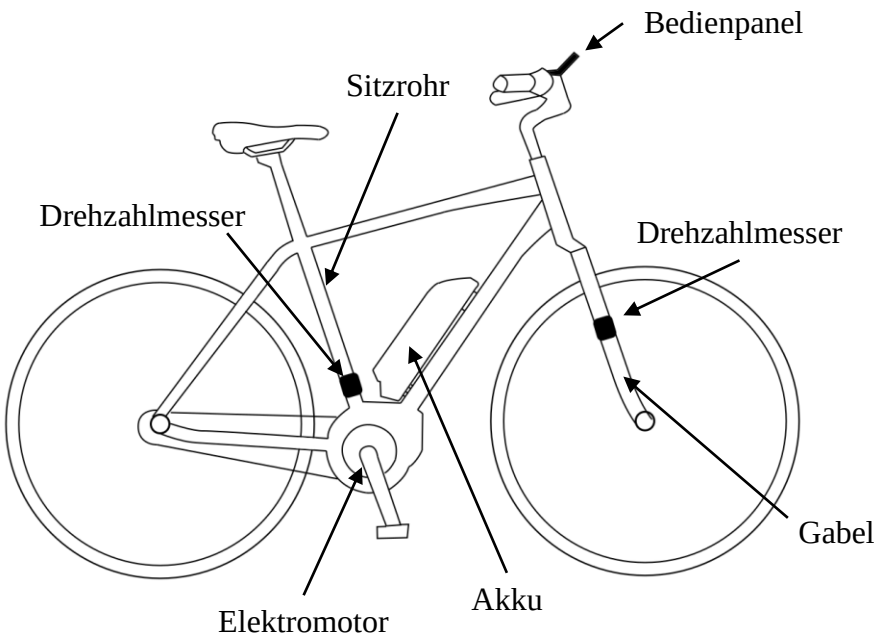


Abb. 17.1 – Sensorik und Aktorik des E-Bikes

Es ist das in **Abb. 17.2** gezeigte Zustandsdiagramm mit den Zustandsnummern 0 bis 5 gegeben. Füllen Sie die durch römische Ziffern gekennzeichneten Lücken im Lösungsfeld aus, bzw. beantworten Sie die gegebenen Fragen.

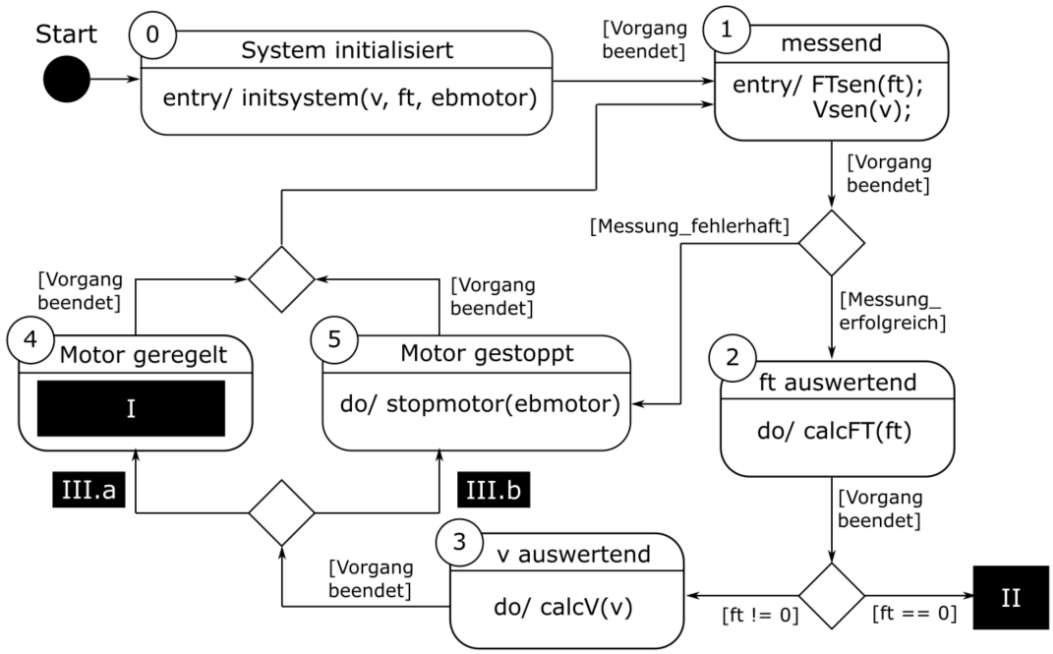


Abb. 17.2 – Zustandsdiagramm der Betriebssoftware des E-Bikes

I. Modellieren Sie den Zustand `Motor geregelt` korrekt aus:

do/ PIDregler(v)

II. Welcher Zustand muss nach dem Zustand `ft auswertend` im Fall `ft==0` folgen, um den Ablauf, wie in der Beschreibung erläutert, zu gewährleisten.

Zustand 5: Motor gestoppt

III. Geben sie die korrekte Wächterbedingung an:

III.a [v<=25]

III.b [v>25]

Aufgabe 18: Zustandsdiagramm zu C-Code

Implementieren Sie Teile des in Aufgabe 17 modellierten Zustandsdiagramms in Form von C-Code. Hierfür stehen Ihnen die in **Tabelle 18.1** dargestellten und bereits implementierten Funktionen zur Interaktion mit den Sensoren und dem Motor, sowie erweiterte Variablen zur Verfügung.

Tab. 18.1 – Sensor- und Aktorvariablen, sowie erweiterte Variablen und bereits implementierte Funktionen

Type	Name	Beschreibung
VARIABLEN	float *v	Zeiger auf die aktuelle Geschwindigkeit.
	float *ft	Zeiger auf die aktuelle Trittfrequenz.
	MOTOR *ebmotor	Zeiger vom Datentyp MOTOR zur Ansteuerung des E-Bike Motors.
	unsigned int vplcZeit	Aktuelle Laufzeit des Programms in Millisekunden.
	unsigned int t	Variable zur timer-Programmierung (zählt in Millisekunden)
	int state	Aktueller Zustand des Zustandsautomaten, welcher ausgeführt wird.
	int mssucv	Variable, in die gespeichert wird, ob die Geschwindigkeit erfolgreich gemessen werden konnte.
	int mssucft	Variable, in die gespeichert wird, ob die Trittfrequenz erfolgreich gemessen werden konnte.
FUNKTIONEN	int Vsen(float *v)	Schreibt die aktuelle Geschwindigkeit und gibt bei erfolgreichem Schreiben eine 1 zurück.
	int FTsen(float *ft)	Schreibt die aktuelle Trittfrequenz und gibt bei erfolgreichem Schreiben eine 1 zurück.
	void initsystem(float *v, float *ft, MOTOR *ebmotor)	Initialisierungsfunktion des Systems.
	void stopmotor (MOTOR *ebmotor)	Anhalten des Motors.

a) Programmgrundgerüst

Vervollständigen Sie das im folgenden Lösungskästchen gezeigte Programmgerüst, um eine zyklische Ausführung des Zustandsautomaten zu ermöglichen. Verwenden Sie hierfür die in **Tabelle 18.1** angegebenen Variablennamen. Zur Interaktion mit der Hardware verwenden Sie die ebenfalls in **Tabelle 18.1** gegebenen Funktionen, welche im Header `ebike.h` bereits definiert und implementiert sind. Die Platzhalter `/* ZUSTAENDE */` enthalten spezifischen Code für Zustände des Zustandsautomaten.

```
//Header fuer Ebike-Interaktion einbinden
  #include "ebike.h"
_____________________________________________
//Variablen fuer Sensoren deklarieren
  float *v, *ft;
_____________________________________________
// Motor
MOTOR *ebmotor;

unsigend int vplcZeit;

int main()
{   // Deklarationen
    int state = 1;        //Startzustand
    int mssucv, mssucft; //Erfolgreiche Messung v, ft
    unsigned int t = 0;  //initiale Zeitvariable

    // initialisiere das System
      initsytem(v, ft, ebmotor);
    ____________________________________________________________

       while(1)
    ____________________________________//Zyklische Ausfuehrung
    {
         switch(state)
        ____________________________________//Zustandsautomat
        {
            /*    ZUSTAENDE   */
            case 5: //halte Motor an (Zustand Motor gestoppt)
                    stopmotor(ebmotor);
                    ___________________________________________
                    state = 1;
                    ___________________________________________
                    break;
                    ___________________________________________

            /*    ZUSTAENDE   */
         }
    }
   return 0;
}
```

b) Implementierung des Zustandes `messend`

Der Zustand `messend` wurde im Folgenden mit einer Zeitbedingung verknüpft und soll nun implementiert werden. Um einzelne Messfehler zu vermeiden, soll mindestens `10 ms` in dem Zustand `messend` verblieben werden. Verwenden Sie vereinfacht für die Zeitmessung die Variable `unsigned int t`, die beim Betreten des Zustandes den Wert 0 hat. Gehen Sie wie folgt vor:
Zu Beginn (nur für t == 0) sollen die Rückgabewerte der Methoden `FTsen` und `Vsen` in die zugehörigen Variablen `mssucft` und `mssucv` gespeichert werden. Nach dem Setzen der Rückgabewerte soll die Zeitmessung starten. Nach Ablauf der `10 ms` soll überprüft werden, ob die Messungen `FTsen` und `Vsen` erfolgreich waren, sodass anschließend in den entsprechenden Folgezustand gewechselt werden soll. Vor Verlassen des Zustandes `messend` soll die Variable `unsigned int t` wieder auf Null zurück gesetzt werden.

```
case 1:     //Zustand messend
_____________________________________
_____________________________________
 if(t == 0)
_____________________________________
 {
_____________________________________
     mssucv = Vsen(v);
_____________________________________
     mssucft = FTsen(ft);
_____________________________________
     t = vplcZeit + 10;
_____________________________________
 {
_____________________________________
 else
_____________________________________
 {
_____________________________________
     if (t < vplcZeit)
_____________________________________
     {
_____________________________________
         t = 0;
_____________________________________
         if (mssucv == 1 && mssucft == 1)
_____________________________________
         {
_____________________________________
             state = 2;
_____________________________________
         }
_____________________________________
         else
_____________________________________
         {
_____________________________________
             state = 5;
_____________________________________
         }
_____________________________________
     }
_____________________________________
 }
_____________________________________
_____________________________________
_____________________________________
_____________________________________
break;
```

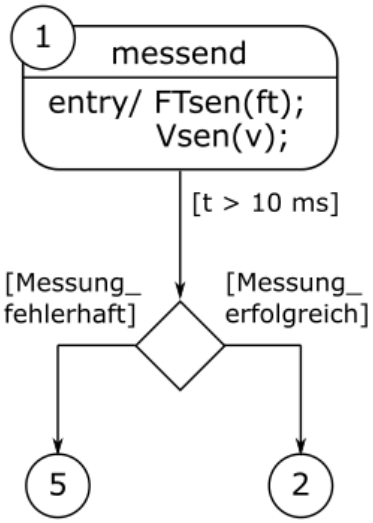


Bild 18.1: *Zustände 1, 5 und 2*

Aufgabe 19: Algorithmen und Datenstrukturen

Das E-Bike soll um eine automatische Gangschaltung ergänzt werden. Dazu werden die 14 zur Verfügung stehenden Gänge durch eine doppelt verkettete Liste im Programmiercode abgebildet.

In einem Listenelement vom Typ `GEAR` soll gespeichert werden:

- Ein Zeiger (`nextgear`), welcher auf das nächste Listenelement vom Typ `GEAR` und somit auf den nächsten Gang zeigt (höherer Gang).
- Ein Zeiger (`prevgear`), welcher auf das nächste Listenelement vom Typ `GEAR` und somit auf den vorherigen Gang zeigt (niedrigerer Gang).
- Die 10 zuletzt gemessenen Trittfrequenzen (`ftarray`) als Fließkommazahl-Array.
- Die maximale Trittfrequenz (`maxtf`) des/der Fahrers/in als Integer.
- Die minimale Trittfrequenz (`mintf`) des/der Fahrers/in als Integer.

Für jeden Gang gibt es eine eigene Motorkonfiguration, die ebenfalls in einer doppelt verketteten Liste gespeichert werden soll.

In einem Listenelement vom Typ `MCONFIG` soll gespeichert werden:

- Ein Zeiger (`nextmc`), welcher auf das nächste Listenelement vom Typ `MCONFIG` und somit auf die nächste Motorkonfiguration zeigt.
- Ein Zeiger (`prevmc`), welcher auf das nächste Listenelement vom Typ `MCONFIG` und somit auf die nächste Motorkonfiguration zeigt.
- Die Nummer der aktuellen Motorkonfiguration (`mode`) als positive Ganzzahl.
- Das maximale Drehmoment (`maxtorque`) des Motors als Ganzzahl.

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

Hinweis: Für alle folgenden Teilaufgaben können Sie annehmen das während des Starts des E-Bikes, durch den/die Fahrer/in die verketteten Listen `GEAR` und `MCONFIG` initialisiert und mit Daten gefüllt werden.

```
typedef struct GEAR {
    struct GEAR *nextgear;
    struct GEAR *prevgear;
    float ftarray[10];
    int maxtf;
    int mintf;
} GEAR;
```

```
typedef struct MCONFIG {
    struct MCONFIG *nextmc;
    struct MCONFIG *prevmc;
    unsigned int mode;
    int maxtorque;
} MCONFIG;
```

b) Im Folgenden soll ein Teil der automatischen Gangschaltung in Form der Funktion `autogearup` modelliert werden. Die Funktion `autogearup` bekommt den aktuellen Gang als Parameter `GEAR *acgear` übergeben. Das Gear-Struct beinhaltet ein Array (`ftarray`) in welchem die letzten 10 Trittfrequenzen abgespeichert sind. Als Rückgabewert (`rtgear`) der Funktion ist, je nachdem ob die Schaltbedingung erfüllt ist, der neue Gang oder der bisherige Gang anzugeben.

Für die Schaltbedingung gilt: Es ist in einen höheren Gang zu schalten, wenn der arithmetische Mittelwert (siehe Gleichung 1) der letzten 10 Trittfrequenzmessungen größer als die maximal zulässige Trittfrequenz (`maxtf`) des jeweiligen Ganges ist.

Zur Berechnung des Mittelwerts ist folgende Formel zu implementieren:

$$f_{aver} = \frac{1}{10}\sum_{i=0}^{9} ftarray_i \qquad \text{GL: 1}$$

Hinweis: Gehen Sie davon aus, dass die Funktion `autogearup` nicht mit dem höchsten Gang aufgerufen wird. Benutzen Sie den Hilfspointer (`frqarray`).

```
GEAR *autogearup(_____GEAR *acgear_____)
{
    float faver = 0; //mittlere Trittfrequenz
    float sum = 0;  // Summe
    GEAR *rtgear;  // return value
    float *frqarray //Hilfspointer

    //Pointer auf ftarray in acgear
    *frqarray = _____acgear->ftarray;_____

    //Berechnung der Summe
    for(_____int i=0; i<10; i++_____)
    {
    _____________________________________________
    ____sum = sum + *(frqarray + i);_____________
    _____________________________________________
    }
    //Berechnung des Mittelwerts
    ___faver = sum/10;___________________________
    _____________________________________________
    ___if(faver > acgear->maxtf)_________________
    ___{_________________________________________
    _____rtgear = acgear->nextgear;______________
    ___}_________________________________________
    ___else______________________________________
    ___{_________________________________________
    _____rtgear = acgear;________________________
    ___}_________________________________________
    _____________________________________________
    return rtgear;
}
```

c) Eine Funktion `runmotor` soll im Folgenden ausgearbeitet werden. Die Funktion erhält als Parameter die Motorkonfiguration vom Typ `MCONFIG`, sowie die aktuelle Trittfrequenz `ft`. Die Funktion `runmotor` berechnet mithilfe der aktuellen Trittfrequenz (`ft`), dem maximalem Drehmoment (`maxtorque`), der Nennfrequenz (`fn`) und der aktuellen Motorkonfiguration (`mode`) das gewünschte Drehmoment (`mtorque`). Hierfür gilt folgende Berechnungsvorschrift (siehe Gleichung 2), die in zwei Schritten (1) und (2) ausgeführt werden soll:

$$m_{torque} = max_{torque} * \underbrace{(1 - \underbrace{\left(\frac{f_t}{f_n} + 1\right)^{-mode}}_{(1)})}_{(2)} \qquad \text{GL: 2}$$

Zuletzt wird das berechnete Moment der Funktion `void tmotor(float *torque)` übergeben.

Hinweis: Für diese Teilaufgabe steht Ihnen die Potenz-Funktion `float powf(float base, float exponent)` zur Verfügung, welche eine Fließkommazahl als Ergebnis zurückliefert.

```
void runmotor (MCONFIG *acmc, float *ft)
{
    float fn = 5;
    float mtorque;
    int maxtorque = ____acmc->maxtorque;_____
    int mode = _____acmc->mode;_____

 //Berechnung (1)_____
    mtorque = powf((*ft)/(fn)+1, (float)-mode);_____
 _____
 _____
 _____
 _____
 _____
 //Berechnung (2)_____
    mtorque = (float) maxtorque * (1 - mtorque);_____
 _____
   tmotor(&mtorque);_____
 _____
 _____
 _____
 _____
}
```

Um Fehler im System zu identifizieren, werden ausgewählte Daten (auch Log-Daten genannt) in dem am Mikrocontroller angeschlossenen EEPROM in Form eines txt-Files gespeichert. Diese können im Fehlerfall durch die Technikerin ausgelesen werden.

d) Implementieren Sie eine Funktion `datalog`, welche das Drehmoment des Motors, (`mtorque`) sowie die zuletzt gemessene Trittfrequenz eines jeden Gangs (insgesamt 14 zur Verfügung stehende Gänge) in einer txt-Datei (`logmt.txt`) ablegt. Die geforderte Trittfrequenz entspricht dabei der letzten Speicherzelle des `ftarray`. Die Werte sollen durch ein Komma getrennt werden. Das File soll dabei nur für den Schreibvorgang geöffnet und danach geschlossen werden. Das Drehmoment ist dabei als Fließkommazahl mit einer Nachkommastelle und die Trittfrequenz als Fließkommazahl mit zwei Nachkommastellen abzuspeichern.

Hinweis: Gehen Sie davon aus, dass der Funktion `datalog` ausschließlich das erste Element der Liste „GEAR“ übergeben wird. Beispiel für einen Datensatz:

20.2,2.34,1.35,1.45,1.28,2.01,1.45,1.28,1.45,1.78,2.21,1.65,1.26,1.93,1.85,

```
void datalog (GEAR *acgear, float *mtorque)
{
   FILE *p = fopen("logmt.txt","w");

   fprintf(p,"%f.1,",*mtorque);

   for (int i= 0; i < 14; i++)
   {
       fprintf(p,"%f.2,", acgear->ftarray[9]);
       acgear = acgear->nextgear;
   }


   fclose(p);


   Alterantive Loesung:
   fprintf(p,"%f.2,", *(acgear->ftarray+9));
}
```