Name: | Vorname: | Matr.–Nr.:

Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

WS 2012/2013
12. Februar 2013

Klausur

Echtzeitsysteme

Aufgabe 1 Wissensfragen (15 Punkte)

1. Erläutern Sie den Unterschied zwischen harten und weichen Echtzeitsystemen. (4 Punkte)
   Lösung:

   - Weiche Echtzeitsysteme:
     Die Berechnungen haben eine zeitliche Ausführungsfrist, eine Überschreitung dieser Fristen hat jedoch keine katastrophale Folgen. Eventuell können die Ergebnisse noch verwendet werden, insgesamt kommt es durch die Fristverletzung evtl. zu einer Dienstverschlechterung.
   - Harte Echtzeitsysteme:
     Eine Verletzung der Berechnungsfristen kann sofort zu fatalen Folgen (hohe Sachschäden oder sogar Gefährdung von Menschenleben) führen. Die Einhaltung der Fristen ist absolut notwendig.

2. Nennen Sie drei Techniken, die bei Standardcomputersystemen zur Performanzsteigerung eingesetzt werden, die jedoch bei *harten* Echtzeitsystemen problematisch sein können. (3 Punkte)

   Lösung:

   - Virtual Memory
   - Caches
   - Superpipelining / Pipelining
   - Garbage Collection
   - Asynchrone I/O
   - ...

Name: | Vorname: | Matr.–Nr.:

---

3. Nennen Sie die jeweiligen Medienzugriffsverfahren, die bei CAN bzw. Ethernet (IEEE 802.3) zum Einsatz kommen. Welches dieser Protokolle ist echtzeitfähig. Begründen Sie Ihre Antwort kurz. (4 Punkte)

   Lösung:

   - CAN: CSMA/CA (alternativ: CSMA/CR)
   - Ethernet: CSMA/CD

   CAN ist bedingt echtzeitfähig (Starvation für niedrigpriore Nachrichten ist möglich), da Kollisionen aufgelöst werden. Ethernet ist aufgrund der zufälligen Backoff-Time nicht deterministisch, und damit nicht echtzeitfähig.

4. Nennen Sie (ohne Erklärung) die vier Kriterien für Verklemmungen (Deadlocks). (4 Punkte)

   Lösung:

   (a) Wechselseitiger Ausschluss (Mutual exclusion)

   (b) Hold-and-wait-Bedingung

   (c) Ununterbrechbarkeit (No preemption)

   (d) Zyklische Wartebedingung (Circular wait)

Name: | Vorname: | Matr.–Nr.:

Aufgabe 2 Scheduling **(25 Punkte)**

Gegeben seien folgende Tasks:

Task T_i	Periode (p_i)	Ausführungszeit (e_i)
T_1	30	10
T_2	45	15
T_3	60	15

Die Tasks sollen nach dem Rate-Monotonic (RM) Algorithmus auf einem Einprozessorsystem ausgeführt werden. Alle Tasks können jederzeit verdrängt werden—sind demnach *preemptable*. Zudem sind die Deadlines der Tasks identisch mit ihrer Periode.

1. Tragen Sie die Prioritäten (hoch, mittel, niedrig) der Tasks in die nachfolgende Tabelle ein. (2 Punkte)

Task T_i	Priorität
T_1	hoch
T_2	mittel
T_3	niedrig

2. Berechnen Sie die *Utilization* des Task-Sets. Ist es möglich auf Basis der berechneten Utilization eine Aussage darüber zu treffen, ob ein *feasible schedule* für die gegebenen Tasks existiert? Zur Hilfe sei die maximale *Utilization* $U_{\max}$ des Schedulability Tests für den RM Algorithmus gegeben: $U_{\max} = 3(2^{1/3} - 1) = 0.7\bar{7}$

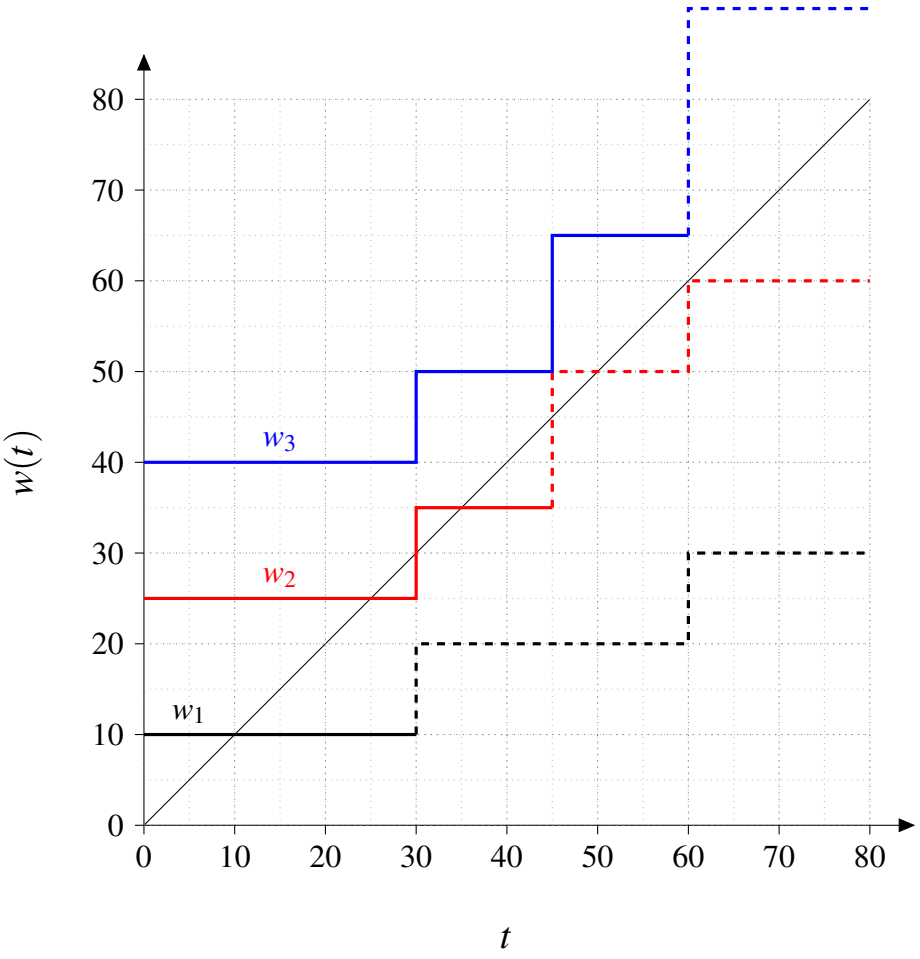
   $U = \sum_{i=1}^{3} \frac{e_i}{p_i} = \frac{10}{30} + \frac{15}{45} + \frac{15}{60} = \frac{11}{12} = 0.91\bar{6}$

   Schedulability Test für RM: $U \leq n \cdot (2^{1/n} - 1) = 3(2^{1/3} - 1) = 0.7\bar{7}$

   Es kann keine Aussage über feasibility getroffen werden, da die Utilization der Tasks größer ist als die maximal zulässige des Schedulability Tests und der Test nur eine *notwendige* aber keine *hinreichende* Bedingung darstellt. (4 Punkte)

3. Führen Sie eine graphische *Time-Demand Analyse* für die gegebenen Tasks durch. Füllen Sie hierzu den nachfolgenden Graphen aus.(6 Punkte)

Name: Vorname: Matr.–Nr.:



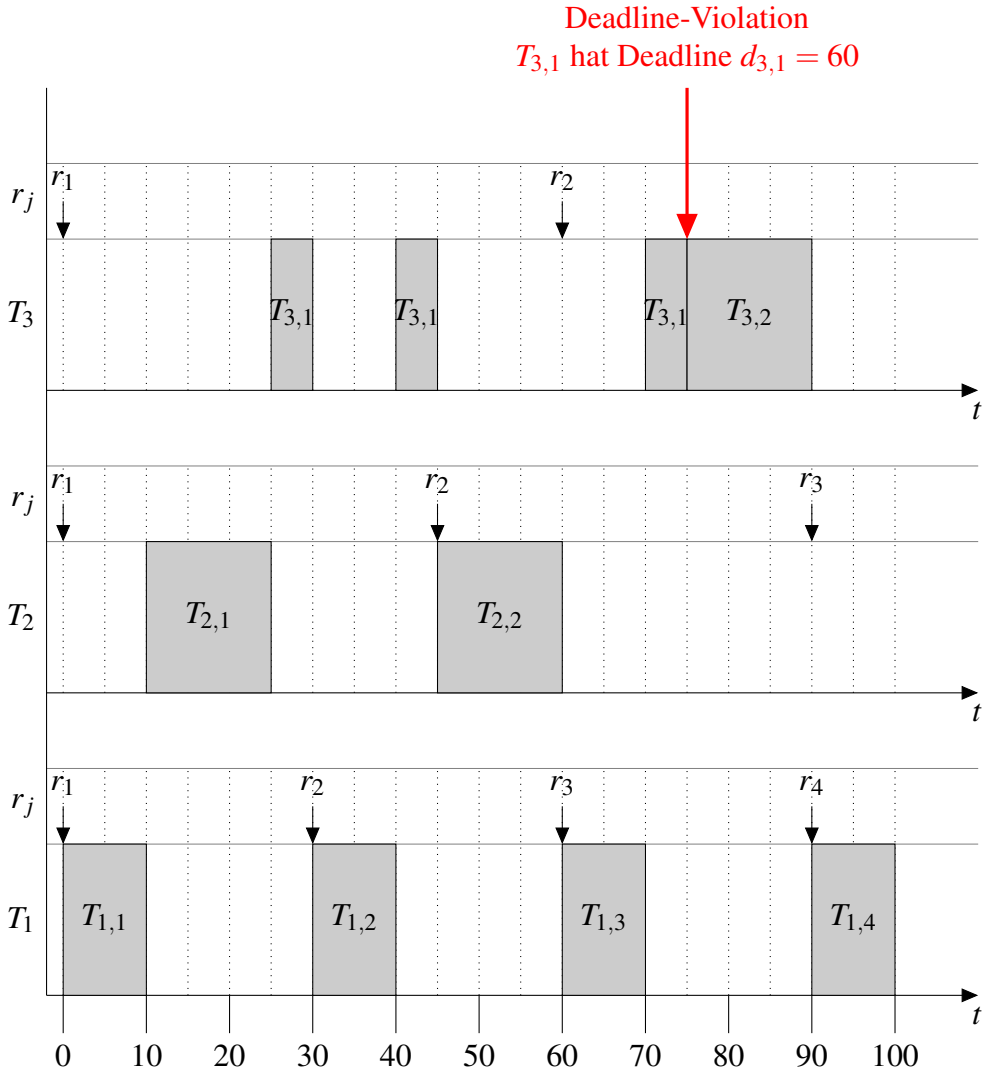
Treten Frist- (Deadline-) Verletzungen auf? Füllen Sie hierzu die nachfolgende Tabelle aus.(2 Punkte)

Task T_i	Fristverletzung (Ja, Nein)
T_1	Nein
T_2	Nein
T_3	Ja

Name: | Vorname: | Matr.–Nr.:

---

4. Zeichnen Sie die resultierende Prozessorbelegung durch die Tasks in die nachfolgende Abbildung ein. Markieren Sie ebenfalls alle Release-Zeitpunkte der Tasks. Beschriften Sie die einzelnen Jobs der Tasks wie folgt (identisch zur Vorlesung): $T_{i,j}$, wobei i die Task-Nummer und j die Jobnummer ist. Die Release-Zeitpunkte der einzelnen Jobs beschriften Sie bitte mit r_j.(8 Punkte)



5. Tragen Sie die *tardiness* der einzelnen Task in die nachfolgende Tabelle ein (3 Punkte):

Task T_i	Tardiness
T_1	0
T_2	0
T_3	15

Name: | Vorname: | Matr.–Nr.:

Aufgabe 3 Nebenläufigkeit **(15 Punkte)**

Mehrere Threads sollen Zugriff auf eine Thread-sichere Queue haben, in der *Items* eingereiht und entnommen werden können. Für die Implementierung steht eine bereits existierende Queue zur Verfügung, die jedoch nicht Thread-sicher ist. Die zu implementierenden Methoden sind *enqueue* zum Einreihen eines *Items* und *dequeue* zum Entnehmen eines *Items*. Diese Methoden sollen durch Semaphoren für nebenläufigen Zugriff geschützt werden. *dequeue* soll blockieren solange kein *Item* in der Datenstruktur vorhanden ist. Nehmen Sie an, dass die Queue eine unbegrenzte Anzahl von *Items* aufnehmen kann. Die nicht Thread-sichere Queue hat folgende Methoden:

```
Queue q;                          ▷ Anlegen einer nicht Thread-sicheren Queue q

Item x;
q.enqueue( x );                   ▷ mit enqueue wird x in der Queue q eingereiht

Item y ← q.dequeue();             ▷ mit dequeue wird ein Item entnommen und hier y zugewiesen
```

Zusätzlich stehen Ihnen Semaphoren mit den folgenden Methoden zur Verfügung

```
Semaphore s;                      ▷ Deklaration einer Semaphore s

s.init( z );                      ▷ initialisiert die Semaphore s mit z

s.up();                           ▷ inkrementiert die Semaphore s

s.down();                         ▷ dekrementiert die Semaphore s und blockiert bis s einen Wert größer 0 hat
```

Implementieren Sie die folgenden Methoden für die Klasse ThreadQueue in Pseudocode und nutzen sie die Methoden der angegebenen Queue und Semaphore. Es kann angenommen werden, dass der Konstruktor *ThreadQueue* vor einem Aufruf von *enqueue* oder *dequeue* einmal aufgerufen worden ist.

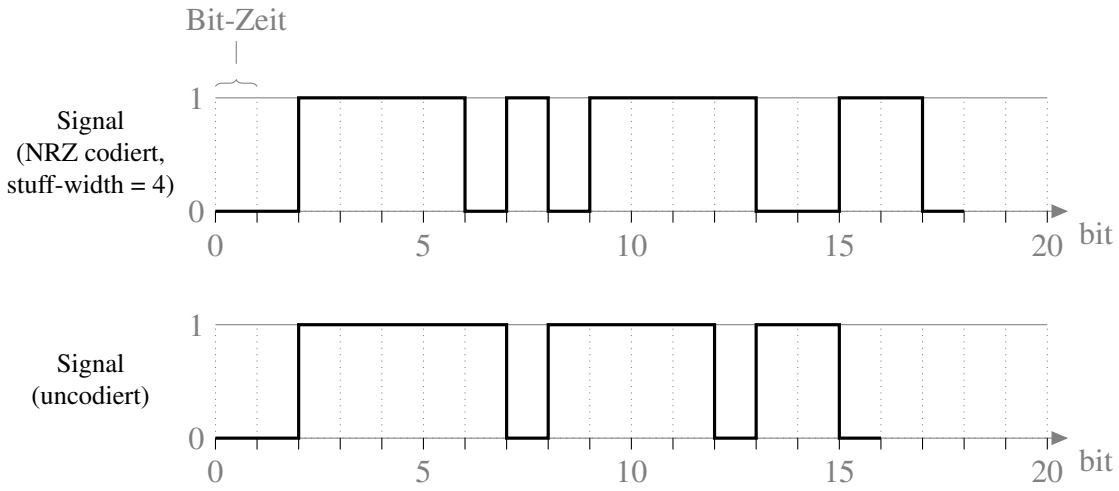
Name: | Vorname: | Matr.–Nr.:

```
 1: class ThreadQueue {
 2:
 3:                                   ▷ Variablen bzw. Semaphoren können hier angelegt werden
 4:     Queue queue;
 5:     Semaphore s;
 6:     Semaphore mutex;
 7:
 8:
 9:     ThreadQueue()
10:     {
11:         s.init( 0 );
12:         mutex.init( 1 );
13:
14:
15:
16:     }
17:
18:     void enqeue( Item x )
19:     {
20:         mutex.down();
21:         queue.enqueue( x );
22:         mutex.up()
23:         s.up();
24:
25:     }
26:
27:     Item deqeue( )
28:     {
29:         s.down();
30:         mutex.down();
31:         Item ret = queue.dequeue();
32:         mutex.up();
33:         return ret;
34:     }
35: }
```

Name: | Vorname: | Matr.–Nr.:

Aufgabe 4 Kommunikation **(20 Punkte)**

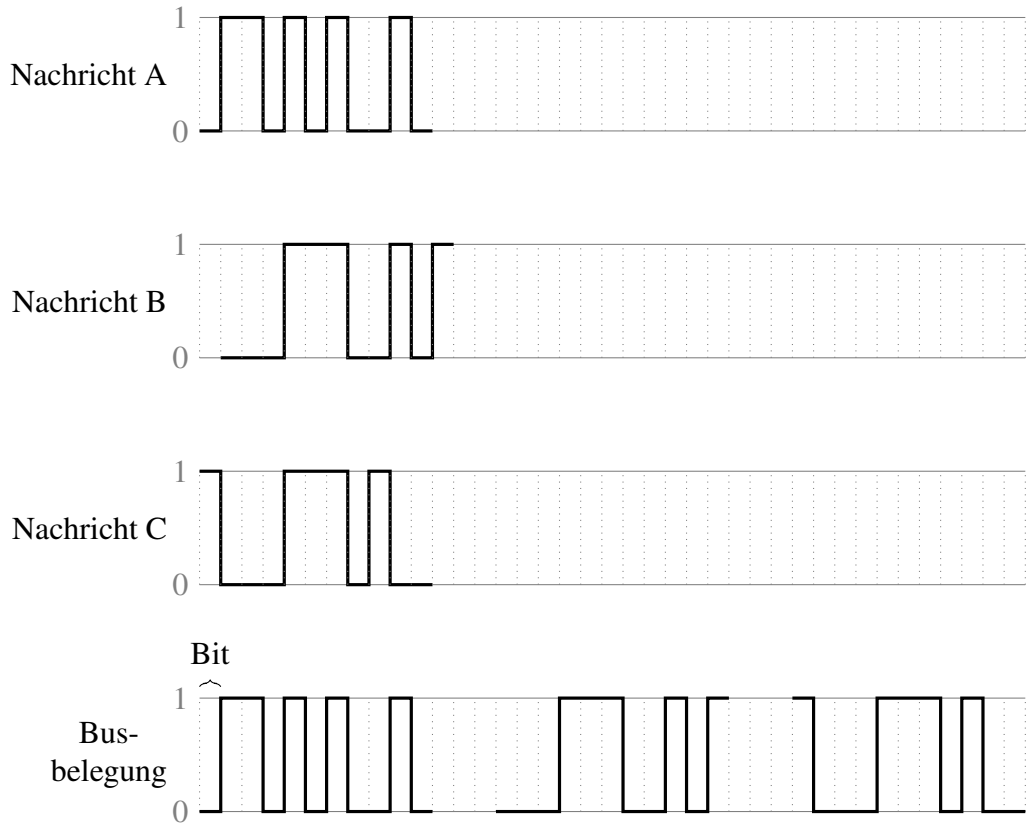
1. Gegeben ist folgende, NRZ-codierte Bitfolge. Dekodieren Sie das Signal und tragen Sie das *Rohsignal* in die Abbildung ein. Der NRZ-Code ist mit einer Stuff-Width von 4 codiert.(3 Punkte)



2. (a) Gegeben seien nun die 11-Bit Identifier von drei CAN-Nachrichten (A, B und C) - siehe nachfolgende Abbildung. Führen Sie auf Basis der gegebenen Identifier eine Arbitrierung der CAN-Nachrichten gemäß des CAN-Protokolls durch und tragen Sie die resultierende Busbelegung in die nachfolgende Abbildung ein. Hierbei soll eine Interframe-Gap von 3 Bit-Zeiteinheiten berücksichtigt werden. Beachten Sie bitte weiterhin folgende Hinweise (10 Punkte):

   - Während der Übertragung des Interframe Gaps ist **kein** Buspegel einzuzeichnen.
   - Außer des CAN-Identifiers werden keine Daten in den CAN-Frames übertragen.
   - Gehe Sie davon aus, dass alle Daten bitsynchron übertragen werden.

Name: Vorname: Matr.–Nr.:



(b) Konvertieren Sie die in (a) gegebenen CAN-Identifier in das Hexadezimalsystem und tragen Sie die konvertierten Identifier in die dafür vorgesehenen Spalte der nachfolgenden Tabelle ein.(5 Punkte)

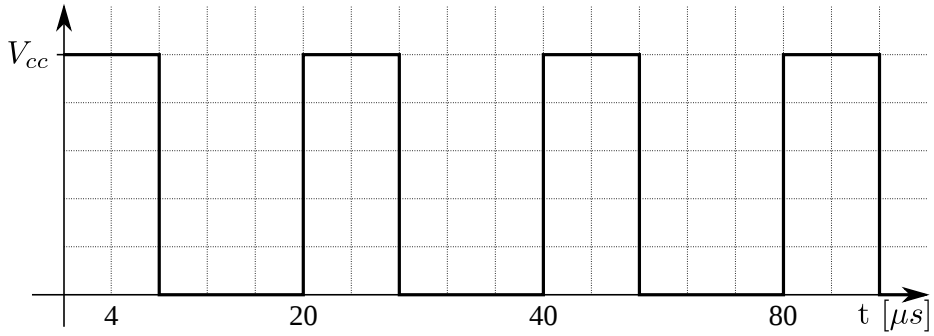
Nachricht	CAN-Identifier (Hexadezimal)
A	0x352
B	0x0E5
C	0x474

(c) Weisen Sie nun den drei CAN-Nachrichten die Prioritäten *niedrig*, *mittel* und *hoch* zu. Füllen Sie hierzu die nachfolgende Tabelle aus (2 Punkte):

Nachricht	Priorität
A	mittel
B	hoch
C	niedrig

Name: | Vorname: | Matr.–Nr.:

Aufgabe 5 Hardware **(15 Punkte)**



1. Gegeben ist ein pulsweitenmoduliertes (PWM) Signal, welches mithilfe eines Timers auf einem Mikrocontroller erzeugt wird. Geben Sie eine mögliche Kombination des Reload Registers mit einem Prescaler an, so dass sich die dargestellte PWM-Frequenz einstellt. Die Prozessorfrequenz beträgt 75 MHz. Für den Prescaler stehen folgende Werte zur Verfügung $p = \{2,3,5,9\}$. (6 Punkte)

$$f_{\text{PWM}} = \frac{1}{20\,\mu\text{s}} = 50\,\text{kHz} \tag{1}$$

$$\frac{75\,\text{MHz}}{50\,\text{kHz}} = 1500 \tag{2}$$

Prescaler $p = \{2,3,5\}$ möglich!

Prescaler	Reload Register
2	$\frac{1500}{2} = 750$
3	$\frac{1500}{3} = 500$
5	$\frac{1500}{5} = 300$

Name: | Vorname: | Matr.–Nr.:

---

2. Geben Sie den *duty cycle* des gegebenen PWM Signals an und errechnen Sie für die im ersten Aufgabenteil berechnete Prescaler-Kombination den entsprechenden Wert des Output Compare Registers. Nehmen Sie an, dass der Timer im *up-counting* Modus betrieben wird. (3 Punkte)

$$\text{dutycycle} = \frac{2}{5} = 0.4 = 40\% \tag{3}$$

Prescaler	Output Compare Register
2	$0.4 \cdot 750 = 300$
3	$0.4 \cdot 500 = 200$
4	$0.4 \cdot 300 = 120$

3. Das Signal eines Kraftsensors soll mit einer Auflösung von mindestens einem Zehntel Newton ausgelesen werden, wobei der Sensor eine lineare Charakteristik mit $g = 50\,\frac{\text{mV}}{\text{N}}$ aufweist. Berechnen Sie die minimal nötige Auflösung in Bits, die der Analog-Digital Wandler bei einer Versorgungsspannung von $V_{cc} = 5\,\text{V}$ aufweisen muss. (6 Punkte)

   (a) Möglichkeit

$$0.1\,\text{N} \cdot 50\,\frac{\text{mV}}{\text{N}} = 5\,\text{mV} \tag{4}$$

$$lsb = \frac{5\,\text{V}}{2^n} < 5\,\text{mV} \tag{5}$$

$$2^n > 1000 \tag{6}$$

$$n \geq 10 \tag{7}$$

   (b) Möglichkeit

$$\frac{lsb}{g} < 0.1\,\text{N} \tag{8}$$

$$\frac{V_{cc}}{2^n \cdot 50\,\frac{\text{mV}}{\text{N}}} < 0.1\,\text{N} \tag{9}$$

$$2^n > \frac{0.1\,\text{N} \cdot g}{V_{cc}} = \frac{0.1\,\text{N} \cdot 50\,\frac{\text{mV}}{\text{N}}}{5\,\text{V}} \tag{10}$$

$$2^n > 1000 \tag{11}$$

$$n \geq 10 \tag{12}$$