

Universität Koblenz–Landau
Institut für Naturwissenschaften
Abteilung Physik

Name: **Musterlösung**
Vorname:
Matr. Nr.:
Studiengang:

Klausur

zur Vorlesung
“Mikrocontroller und Robotik”

Mittwoch 5.8.2009

Lösen Sie die Aufgaben 1 - 6!
Verwenden Sie keinen Bleistift!
Es sind keine Hilfsmittel zugelassen.
Schalten Sie Ihr Handy aus!

GUTEN ERFOLG !!!

Aufgabe	1	2	3	4	5	6
max. Punkte	8	12	12	10	10	8
err. Punkte						

Summe Pkte.:

Note.:

Aufgabe 1: Diskrete Treiberschaltungen

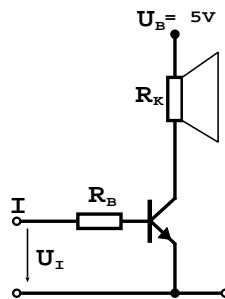
Ein Transistor verhält sich wie ein regelbarer Widerstand. Dabei steuert der Basisstrom I_B den Kollektorstrom I_C . Beide Ströme stehen in einem festen Verhältnis zueinander:

$$B = \frac{I_C}{I_B}$$

Dieses Verhältnis wird Stromverstärkungsfaktor "B" genannt, der eine typische Kenngröße von Transistoren darstellt, und in jedem Datenblatt nachgeschlagen werden kann.

Die Basis eines Transistors ist sehr empfindlich und kann bereits mit relativ kleinen Strömen zerstört werden. Der Basisstrom ist deshalb so gering wie möglich, aber gleichzeitig so groß wie nötig einzustellen.

Gegeben ist nun die Stromverstärkungsschaltung aus unserem WAV-Player:



Die Eingangsspannung U_I beträgt maximal $U_B = 5V$. Der Ohm'sche Widerstand des Kopfhörers R_K sei 50Ω . Der Stromverstärkungsfaktor B sei 10.

- a) Berechnen Sie den Basisvorwiderstand R_B , sodass der Kopfhörer maximal angesteuert wird, bei gleichzeitig minimalem Basisstrom

Hinweis:

I_B ist nur von U_I und R_B abhängig

I_C ist nur von B und I_B abhängig

Der maximale Strom, der durch den Kopfhörer fließen kann, wird durch dessen Innenwiderstand bestimmt:

$$I_{R_K} = I_C = \frac{U_0}{R_K} = \frac{5V}{50\Omega} = 0,1A = 100mA$$

Damit ein Kollektorstrom von $100mA$ fließen kann, benötigen wir einen Basisstrom von:

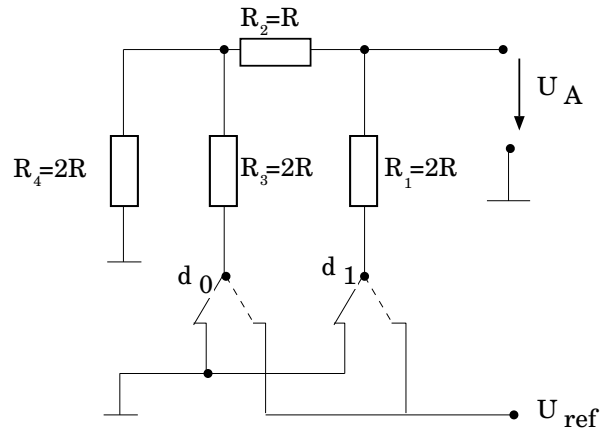
$$B = \frac{I_C}{I_B} \Leftrightarrow I_B = \frac{I_C}{B} = \frac{100mA}{10} = 10mA$$

Um einen Basisstrom von $10mA$ einzustellen, brauchen wir bei $U_I = 5V$ einen Basisvorwiderstand von 500Ω

() / 8

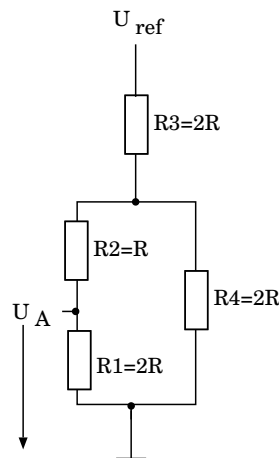
Aufgabe 2: DA-Wandler

Gegeben sei folgender 2-Bit R2R-DA-Wandler:



Berechnen Sie die Ausgangsspannung U_A für die Belegung $d_1 = 0$ und $d_0 = 1$. U_{ref} sei 16 Volt, der Widerstand R beträgt $1\text{ k}\Omega$

a) Zeichnen Sie ein äquivalentes Ersatzschaltbild für die gegebene Belegung.



() / 1

b) Berechnen Sie den Gesamtwiderstand

$$\begin{aligned}
 R_g &= R_3 + (R_1 + R_2) || R_4 \\
 &= 2R + 3R || 2R \\
 &= 2R + \frac{2 * 3}{2 + 3} R \\
 &= \frac{16}{5} R = \frac{16}{5} \text{ k}\Omega
 \end{aligned}$$

() / 2

c) Berechnen Sie den Gesamtstrom und die entsprechenden Teilströme

$$\begin{aligned}
 I_g &= U_{ref}/R_g \\
 &= \frac{16V}{\frac{16}{5}k\Omega} = 5mA \\
 I_{R_3} &= 5mA \\
 I_{R_{1+2}} + I_{R_4} &= I_{R_3} = 5mA \\
 I_{R_4} &= 3mA \quad da \quad R_4 = 2k\Omega \\
 I_{R_{1/2}} &= 2mA \quad da \quad R_{1+2} = 3k\Omega
 \end{aligned}$$

() / 4

d) Berechnen Sie den Spannungsabfall an allen Widerständen

$$\begin{aligned}
 U_{R_3} &= R_3 * I_{R_3} = 2k\Omega * 5mA = 10V \\
 U_{R_4} &= R_4 * I_{R_4} = 2k\Omega * 3mA = 6V \\
 U_{R_1} + U_{R_2} &= U_{R_4} = 6V \\
 U_{R_1} &= R_1 * I_{R_{1+2}} = 2k\Omega * 2mA = 4V \\
 U_{R_2} &= R_2 * I_{R_{1+2}} = 1k\Omega * 2mA = 2V
 \end{aligned}$$

() / 4

e) Bestimmen Sie die Ausgangsspannung U_A . Überprüfen Sie Ihr Ergebnis anhand der bekannten Formel zur Berechnung der Ausgangsspannung am R2R-Wandler.

$$\begin{aligned}
 U_A &= U_{R_1} = 4V \\
 U_A &= D \frac{U_{ref}}{2^n} = 1 \frac{16V}{4} = 4V
 \end{aligned}$$

() / 1

Aufgabe 3: Stack

- a) Was ist der Stack (bzw. der Stackpointer) und wozu wird er gebraucht?

Der Stack ist ein Bereich des **RAMs**, der durch den Stackpointer referenziert wird. Dieser Bereich realisiert einen **Kellerspeicher** (oder **LIFO-Prinzip**). Jeder **Prozess reserviert** einen eigenen Stack. Die eigentliche Verwaltung des Stacks erfolgt im Prozessor auf **Hardwareebene**. Der Stackpointer ist ein **Prozessorregister** mit eigenen **Increment-** und **Decrementeinheiten**. Es existieren eigene Assemblerbefehle **PUSH** und **POP** zur Verwaltung des Stacks. Der Stack wird genutzt zum Aufruf von **Unterprogrammen**, zur **Parameterübergabe/rückgabe**, zur **Rettung von Registerinhalten** und für lokale Variablen

() / 4

- b) Was passiert genau bei CALL und RETURN?

CALL: Der CALL Befehl packt den bereits incrementierten Programmcounter auf den Stack und decrementiert anschließend des Stackpointer. Der Programmcounter bekommt die Einsprungadresse des Unterprogramms zugewiesen:

$$*(SP - -) = ++ PC$$

$$PC = UPA\text{dresse}$$

RETURN Zunächst wird der Stackpointer incrementiert. Der Return Befehl weist anschließend dem Programmcounter den nun aktuellen Wert des Stacks zu.

$$PC = *(++ SP)$$

() / 4

- c) Worin liegt bei der Assemblerprogrammierung der kleine aber notwendige Unterschied zwischen einem Unterprogrammaufruf(CALL) und dem Aufruf einer ISR (Interrupt Service Routine) bzgl. der durchzuführenden Stackoperationen?

Neben der Rettung aller benutzten Register muss bei einer ISR zusätzlich das Statusregister gerettet werden.

() / 2

- d) Begründen Sie diesen Unterschied.

Da der Aufruf einer ISR von der Hardware erzeugt wird, ist es in einem Programm nicht vorhersagbar, wann ein solcher Aufruf erfolgt. Es ist somit möglich, dass der Aufruf einer ISR zwischen einer arithmetisch-logischen Operation und der nachfolgenden bedingten Sprunganweisung erfolgt. Da die ISR mit Sicherheit das Statusregister beeinflusst, würde die bedingte Sprunganweisung aufgrund eines verfälschten Statusregister entschieden.

Das Programmverhalten wäre nicht mehr nachvollziehbar.

() / 2

Aufgabe 4: AVR-Assembler

Ihr Partner stellt Ihnen folgendes Unterprogramm (mit kleinem Testprogramm) zur Verfügung, das eine 16 Bit Delayschleife implementiert:

```
Adr      Befehl
-----
0000:    ldi r16,high(RAMEND)
0001:    out SPH,r16
0002:    ldi r16,low(RAMEND)
0003:    out SPL,r16          ; init stack

0004:    ldi r16,0x04        ; test with
0005:    push r16            ; only 4 loops
0006:    ldi r16,00         ; LSB First
0007:    push r16            ; then MSB
0008:    call delay
000A:    :

;-----
; 16 Bit delay Loop
; Parameter is read from stack, so
; push paramter on stack before calling delay
;-----
delay:
0100:    pop XH              ; pop parameter (LIFO)
0101:    pop XL              ; first MSB then LSB
      LO:
0102:    sbiw X,1           ; decrement X (16 bit operation)
0103:    brne LO            ; until X=0
0104:    ret                ; then return
```

Leider funktioniert das Programm nicht wie erwartet. Keine der Befehle die im Hauptprogramm dem Call-Befehl folgen werden mehr ausgeführt. (Es sind keine syntaktischen Fehler vorhanden!):

- a) Analysieren Sie das Programm. Welchen Fehler hat Ihr Partner gemacht?

Das Unterprogramm interpretiert die Rücksprungadresse als Parameter und den Parameter als Rücksprungadresse.

() / 2

b) Wie wird sich das Programm nach einem Start verhalten?

Da die Rücksprungadresse als Parameter interpretiert wird, werden die Schleifen nicht 0x0004 mal durchlaufen, sondern 0x000A mal. Zudem wird nach Ablauf des Unterprogramms nicht die korrekte Rücksprungadresse 0x000A angesprungen, sondern die Adresse 0x0004. An dieser Stelle wird der 16 Bit Parameter neu auf den Stack gepackt und anschliessend das Unterprogramm erneut aufgerufen.

Das Programm befindet sich somit in einer Endlosschleife.

() / 6

c) Wie muss das Programm modifiziert werden, dass es sich, wie im Kommentar beschrieben, verhält? (Sie können beliebige Register benutzen)

```
delay:  pop r17          ; save return address
        pop r18
        pop XH          ; pop Parameter
        pop XL
        push r18         ; restore return address
        push r17
L0:     sbiw X,1         ; decrement X
        brne L0         ; until X=0
        ret              ; then return
```

() / 2

Aufgabe 5: C - Programmierung

Interpretieren Sie folgende C-Funktion:

```
void function(int a, int b)
{
    a = a ^ b;
    b = a ^ b;
    a = a ^ b;
}
```

Hinweis: \wedge = XOR

- a) Welche Operation führt die Funktion auf den beiden Variablen a und b aus? (Welche Inhalte haben die beiden Variablen am Ende der Funktion?)

Zeile1: $a = a \wedge b$;

Zeile2: $b = a \wedge b = (a \wedge b) \wedge b = a$

Zeile3: $a = a \wedge b = (a \wedge b) \wedge a = b$

Die Funktion tauscht die Inhalte der beiden Variablen: SWAP-Funktion

() / 4

- b) Welchen Fehler hat der Programmierer gemacht?

Da die Parameter *call by value* übergeben werden, behalten Sie im aufrufenden Program ihre Inhalte. Die Funktion ist somit nutzlos. Die Parameter müssen *call by reference* übergeben werden.

() / 2

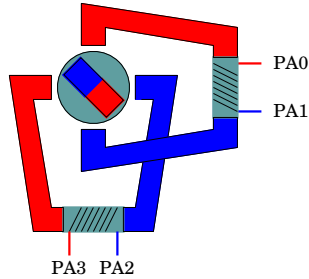
- c) Notieren Sie eine korrekte Version der Funktion.

```
void function(int *a, int *b)
{
    *a = *a ^ *b;
    *b = *a ^ *b;
    *a = *a ^ *b;
}
```

() / 4

Aufgabe 6: C Programmierung

Entwerfen Sie ein C-Programm zur Ansteuerung eines Schrittmotors. Der Schrittmotor soll im **Halbschritt** betrieben werden.



- a) Geben Sie zunächst die logische Steuerfolge (in 1/0 Notation) für den **Halbschrittbetrieb** in beliebiger Drehrichtung an:

Schritt Nr.	PA3	PA2	PA1	PA0
0	0	1	0	1
1	0	1	0	0
2	0	1	1	0
3	0	0	1	0
4	1	0	1	0
5	1	0	0	0
6	1	0	0	1
7	0	0	0	1

() / 4

- b) Schreiben Sie ein möglichst kurzes C-Programm zur Ansteuerung. Verwenden Sie PortA als Ausgabe. Eine Konfiguration des Ports ist nicht gefordert, ebensowenig wie Delays zwischen den Ausgaben.

```
int main()
{
    unsigned char i,motor[8]={0b0101,0b0100,0b0110,0b0010,
                              0b1010,0b1000,0b1001,0b0001};
    while(1)
        PORTA=motor[i++%8];
}
```

() / 4