Vorbemerkungen:

Die folgenden Beispiele beziehen sich auf das Uno-Modul-Shield mit aufgestecktem **Arduino-Uno** oder **Arduino-Nano**. Es stehen drei Ports mit unterschiedlicher Anzahl Bits zur Verfügung (Abb. 1).

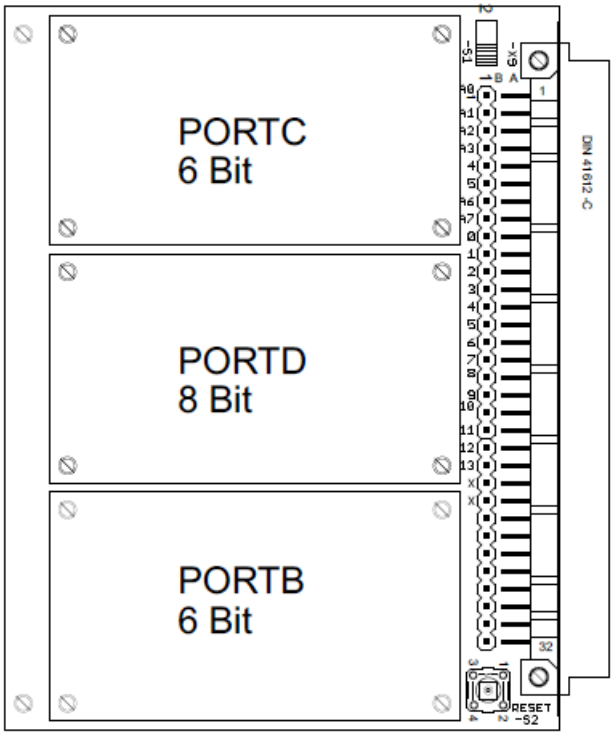

Abb. 1: Uno-Modul-Shield ohne Module

Auf die Ports können folgende Module aufgesteckt werden:

- **Switch-Modul**
- **Poti-Modul**
- **LED-Modul**
- **7Segm-Modul**
- **LCD-Modul**

Weiterhin können an die 64pol. Messerleiste Großmodule (z.B. Wetterstation, Codierer, LED-Cube) angesteckt werden.

In diesem Skript wird die integrierte Entwicklungsumgebung **AVR-Studio 6.2** mit dem Arduino-Add-On **Visual Micro** benutzt. Der Hauptvorteil im Vergleich zur **Arduino-IDE** ist der verbesserte Editor, so gibt es beispielsweise Intellisense, ein Feature zur automatischen Quelltext-Vervollständigung. Bei der Installation geht man folgendermaßen vor:

- zuerst muss die Arduino-IDE installiert werden, diese findet man hier: https://www.arduino.cc/en/Main/Software
- dann installiert man das AVR-Studio 6.2 unter: http://www.atmel.com/tools/atmelstudio.aspx
- zuletzt das Arduino-Add-On installieren, dieses befindet sich hier: http://www.visualmicro.com/page/Arduino-Visual-Studio-Downloads.aspx

Die Beispiele befinden sich jeweils in einem separaten Ordner, z.B. in **bsp_01_zahl**. Zum Öffnen der Beispiele genügt es, auf die entsprechende Solution-Datei (*.atsln) zu klicken, dann öffnet sich das Projekt automatisch. Dies funktioniert natürlich nur bei installiertem AVR-Studio mit Arduino-Add-On. Benutzt man die einfache Arduino-IDE, so muss man die entsprechende *.ino-Datei laden, diese befindet sich im gleichnamigen Unterverzeichnis innerhalb der Beispielordner. Der Download auf das Arduino-Board erfolgt dank des vorhandenen Bootloaders ohne spezielles Programmiergerät direkt über USB.

Übersicht der Beispiele:

bsp_01_zahl
bsp_02_schaltertest
bsp_03_blink
bsp_04_blink
bsp_05_if_else
bsp_06_if_else_if
bsp_07_switch_case
bsp_08_count
bsp_09_lauflicht
bsp_10_bargraph
bsp_11_array
bsp_12_gray
bsp_13_taktmerker
bsp_14_LED_multiplex
bsp_15_stoppuhr
bsp_16_LCD
bsp_17_AD_Wandler
bsp_18_LCD_Voltmeter
bsp_19_LED_Voltmeter
bsp_20_Encoder
bsp_21_Encoder_LCD
bsp_22_Encoder_7segm

Hinweis zur Softwaresprache:

In diesem Skript werden Arduino-Spezialfunktionen nur dort verwendet, wo es Sinn macht. Ansonsten wird soweit wie möglich in ANSI-C programmiert. Die Verwendung vieler Arduino-Funktionen erzeugt einen großen Code-Overhead im Programmspeicher des verwendeten Mikrocontrollers. Die häufige Verwendung vorgefertigter Arduino-Funktionen ist außerdem kein Industriestandard, wo Quelltexte nicht zuletzt wegen der Portierbarkeit auf andere µC-Systeme weitgehend in ANSI-C geschrieben sind.

In der Arduino-Welt gibt es die beiden Funktionen **`setup()`** und **`loop()`**. In der Funktion **`setup()`** stehen Anweisungen, die genau einmal abgearbeitet werden müssen, z.B. die Einstellung der Portrichtungsregister (DDRx). Die Funktion **`loop()`** enthält den Hauptzyklus, der periodisch abgearbeitet wird. Diese beiden Funktionen werden im gesamten Skript verwendet, sie bilden das Gerüst eines jeden Quelltextes. In ANSI-C werden diese beiden Funktionen von der Hauptfunktion **`main()`** gebildet, in welcher der Hauptzyklus innerhalb einer Endlosschleife abgearbeitet wird.

Um die Großmodule (z.B. Codierer, Wetterstation, LED-Cube) zu programmieren, sollten zunächst die Beispiele und Aufgaben dieses Skripts bearbeitet werden. Die Großmodule werden jeweils in einem extra Skript mit Beispielen und Aufgaben beschrieben.

1 bsp_01_zahl

Im Beispiel werden verschiedene Zahlen auf die Ports geschrieben und das Ergebnis beobachtet.

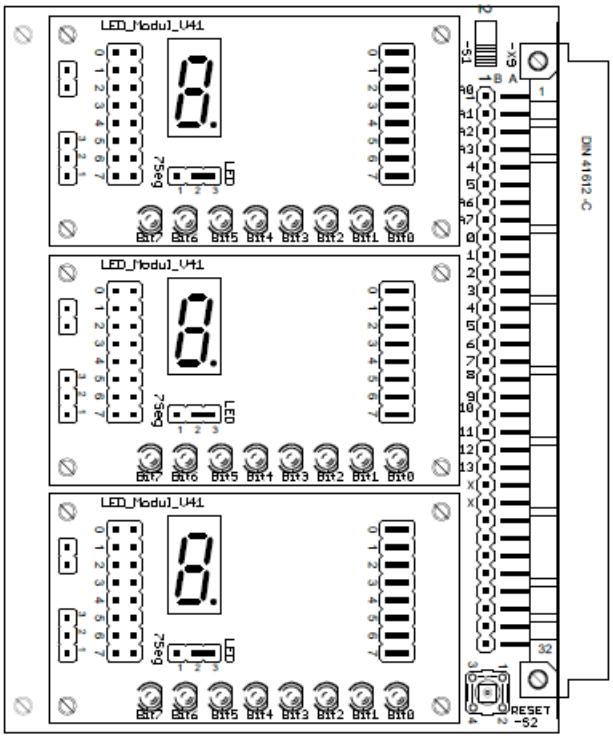

Abb. 2: Konfiguration der Module

```
17  void setup() {
18
19  // PORTx:    1=output
20  //           0=input
21       DDRC=0x3F;  // PORTC 6 Bit
22       DDRD=0xFF;  // PORTD 8 Bit
23       DDRB=0x3F;  // PORTB 6 Bit
24  }
25
26  void loop() {
27
28       PORTC=0x35;      //  Hexzahl
29       PORTD=9;         //  Dezzahl
30       PORTB=0b110011; //  Binzahl
31  }
32
```

Abb. 3: Ausschnitt aus dem Quelltext bsp_01_zahl

Das Uno-Modul-Shield sollte gemäß obenstehender Abbildung konfiguriert werden (Jumper beachten). Falls nicht 3 LED-Module vorhanden sind, können die Ports auch nacheinander beobachtet werden durch Umstecken eines einzigen LED-Moduls.

In der Funktion **`setup()`** werden die Portrichtungsregister (DDRx) eingestellt. Da hier nur Ausgänge vorliegen, erhalten alle Bits eine 1 (vgl. Kommentare in Abb. 3).
In der Funktion **`loop()`** werden die drei Ports mit Zahlen beschrieben, die mit den LEDs beobachtet werden können. Man könnte die Anweisungen in den Zeilen 28 bis 30 auch in die Funktion **`setup()`** schreiben, da sich im Zyklus hier nichts ändert. Die Funktion **`loop()`** bleibt dann leer.

<u>Aufgaben:</u>

1 Welches ist die größte Zahl, die in ein 8-Bit-Register (z.B. Ports) hineinpasst ? Geben Sie diese Zahl binär, hexadezimal und dezimal an!

2 Wandeln Sie die Dezimalzahlen **56**, **178** und **203** in Binär- und in Hexadezimalzahlen um. Übertragen Sie dann die gefundenen Hexzahlen auf die drei Ports und vergleichen Sie das Ergebnis mit der jeweiligen Binärdarstellung !

2 bsp_02_schaltertest

Im Beispiel werden die Schalterstellungen an Port C auf Port D kopiert, wobei der Port C nur 6 Bit hat.

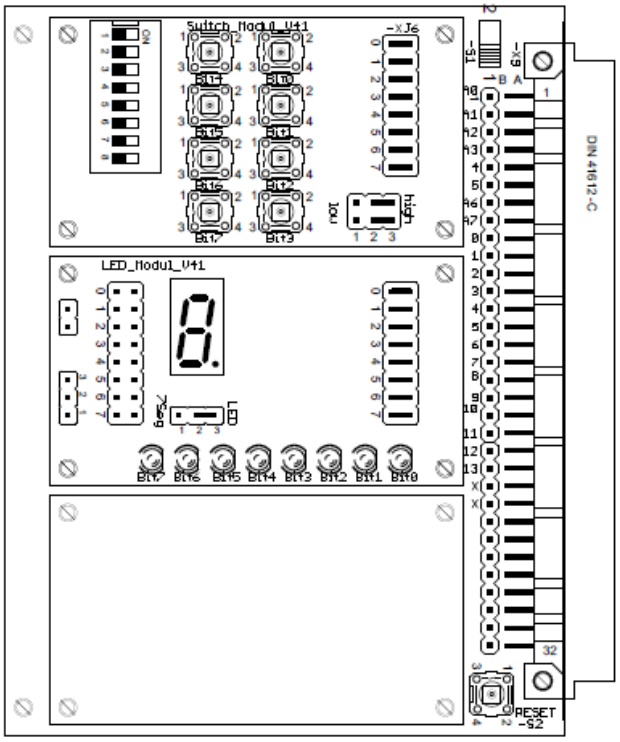

Abb. 4: Konfiguration der Module

```
15    //----------------------
16      void setup() {
17
18      // PORTx:    1=output
19      //           0=input
20      DDRC = 0x00;
21      PORTC= 0x00;
22      DDRD = 0xFF;
23      DDRB = 0xFF;
24      }
25    //----------------------
26      void loop() {
27
28      PORTD=PINC;
29      }
30    //----------------------
```

Abb. 5: Ausschnitt aus dem Quelltext bsp_02_schaltertest

Das Uno-Modul-Shield sollte gemäß obenstehender Abbildung konfiguriert werden (Jumper beachten). In der Funktion **`setup()`** werden die Portrichtungsregister (DDRx) eingestellt. Ausgangsbits erhalten eine 1, Eingangsbits eine 0 (vgl. Kommentare in Abb. 5).
In der Funktion **`loop()`** wird nun der Zustand von PINC auf den PORTD übertragen. Beim Abfragen eines Eingangs muss PINx benutzt werden. Beim Betätigen eines Schalters leuchtet das betreffende Bit auf.

Aufgaben:

1 Setzen Sie die beiden Jumper auf dem Switch-Modul von *high* nach *low* und testen Sie das Ergebnis aus!

2 Wechseln Sie die Ports. Machen Sie den Port B zum Eingangsport und den Port C zum Ausgangsport! Testen Sie die Ergebnisse aus!

3 bsp_03_blink

Im Beispiel wird Bit 1 des Port D blinken mit einer Blinkfrequenz von 0,5 Hz.

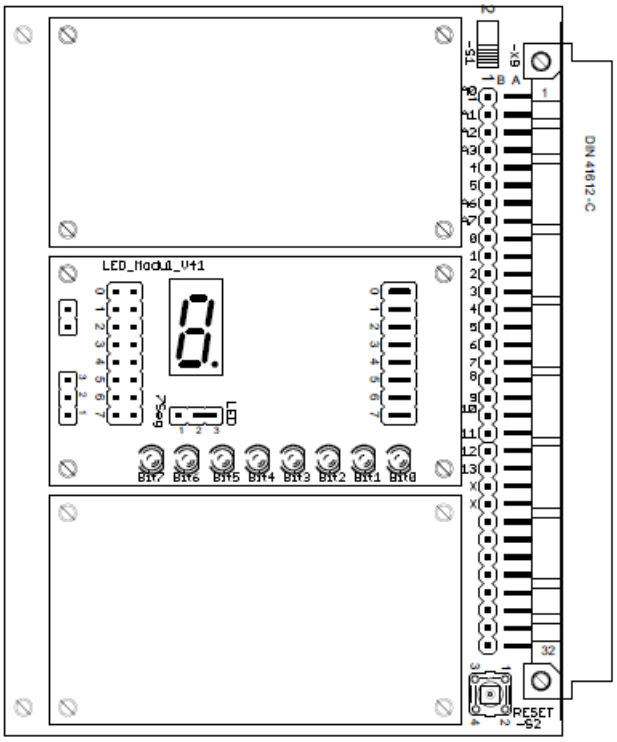

Abb. 6: Konfiguration der Module

```
10  //----------------------------
11      #include <avr/io.h>
12      #define F_CPU 16000000UL
13      #include <util/delay.h>
14
15  //----------------------------
16      void setup() {
17
18      DDRC = 0x00;
19      PORTC= 0x00;
20      DDRD = 0xFF;
21      DDRB = 0xFF;
22      }
23  //----------------------------
24      void loop() {
25
26      PORTD |=0b00000010;
27      _delay_ms(1000);
28      PORTD &=0b11111101;
29      _delay_ms(1000);
30      }
31  //----------------------------
```

Abb. 7: Ausschnitt aus dem Quelltext bsp_02_schaltertest

In Zeile 11 wird die AVR-Headerdatei eingebunden, dort sind Port- und Registernamen definiert. In Zeile 12 wird der CPU-Takt (16 MHz) definiert, in Zeile 13 wird die Headerdatei für die Zeitverzögerungen eingebunden. In Zeile 26 wird Bit 1 des Port D gesetzt durch bitweise OR-Verknüpfung, in Zeile 28 wird das Bit wieder gelöscht durch bitweises AND. Die Zeilen 27 und 29 enthalten die Zeitverzögerungen.

4 bsp_04_blink

Hier blinkt Bit 2 mit einer Frequenz von 0,5 Hz durch bitweise Anwendung der ExOr-Verknüpfung. Die Konfiguration der Module ist die gleiche wie in bsp_03_blink.

```
22  //-------------------------
23      void loop() {
24
25          PORTD=PORTD^0x04;
26          _delay_ms(1000);
27
28      }
29  //-------------------------
```

Abb. 8: Ausschnitt aus dem Quelltext bsp_04_blink

Aufgaben:

1 Lassen Sie die Bits 3 und 5 blinken nach der Methode von bsp_03_blink!

2 Benutzen Sie nun die Methode von bsp_04_blink für die Bits 0 und 4 ! Ändern Sie auch die Zeiten und die Ports!

5 bsp_05_if_else

Das Beispiel zeigt eine zweifache Auswahl. Es sollen die Bits 0 bis 3 des Port D leuchten, wenn die Taste an Bit 1 des Port C betätigt wird. Wird nichts betätigt, sollen die Bits 4 bis 7 des Port D leuchten.

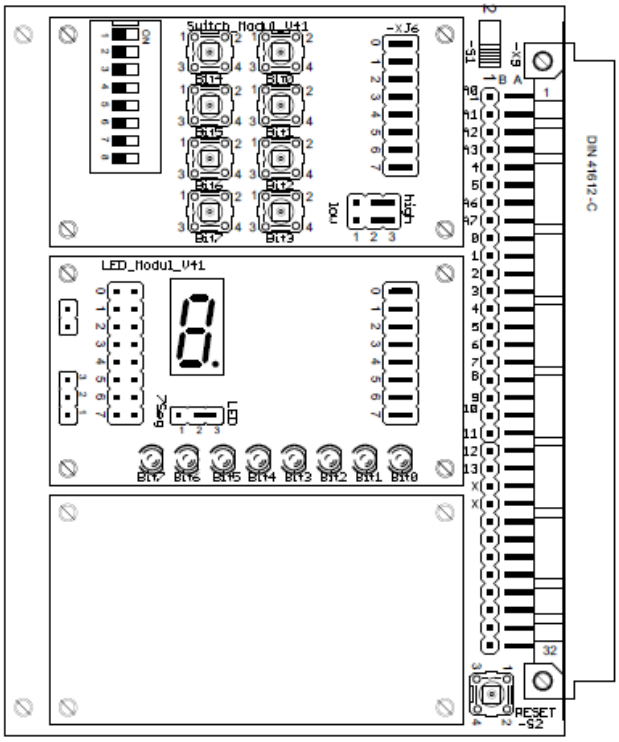

Abb. 9: Konfiguration der Module

Das Uno-Modul-Shield wird gemäß nebenstehender Abbildung konfiguriert (Jumper beachten). In der Funktion **`setup()`** werden die Portrichtungsregister (DDRx) eingestellt (Abb. 10).

```
17  //--------------------------------
18      void setup() {
19           DDRC = 0x00;
20           PORTC= 0x00;
21           DDRD = 0xFF;
22           DDRB = 0xFF;
23      }
24  //--------------------------------
25      void loop() {
26
27           if(PINC&0x01) PORTD=0x0F;
28           else PORTD= 0xF0;
29      }
30  //--------------------------------
```

Abb. 10: Ausschnitt aus dem Quelltext bsp_05_if_else

Die Zeile 27 in Abb. 10 enthält die Abfrage nach Bit 1 des Port C. Wenn der Klammerausdruck wahr ist, dann werden die unteren 4 LEDs an Port C leuchten, bei falschem Ergebnis in der Klammer leuchten gemäß Zeile 28 die oberen 4 Bits des Port D.

Aufgaben:

1 Wenn die Taste an Bit 3 des Port C betätigt wird, sollen die LEDs an den Bits 0, 2, 4, 6 des Port D leuchten, ansonsten sollen die übrigen LEDs des Port D leuchten !

2 Die LEDs an den Bits 3 und 4 des **Port B** sollen leuchten, wenn die Taste an Bit 5 des **Port D** betätigt wird, ansonsten sollen die LEDs an den Bits 0, 1 und 2 leuchten (Achtung: Board muss umkonfiguriert werden) !

6 bsp_06_if_else_if

Das Beispiel zeigt eine **mehrfache Auswahl**. Die Konfiguration des Uno-Modul-Shields ist identisch mit der von Beispiel 5. Abhängig davon, welche Taste des Port C betätigt wird, soll an Port D die entsprechende LED leuchten. Wenn keine Taste betätigt wird, sollen die LEDs an den Bits 4 bis 7 leuchten.

```
27  //-------------------------------------
28     void loop() {
29
30          if(PINC&0x01) PORTD=0x01;
31          else if (PINC&0x02) PORTD=0x02;
32          else if (PINC&0x04) PORTD=0x04;
33          else if (PINC&0x08) PORTD=0x08;
34               else PORTD= 0xF0;
35     }
36
37  //-------------------------------------
```

Abb. 11: Ausschnitt aus dem Quelltext bsp_06_If_else_if

Taste an Bit 0 betätigt → LED an Bit 0 des Port D leuchtet,
…
Taste an Bit 3 betätigt → LED an Bit 3 des Port D leuchtet
Keine Taste betätigt → LEDs an den Bits 4 bis 7 leuchten

7 bsp_07_switch_case

Die Funktion in diesem Beispiel ist ebenfalls eine mehrfache Auswahl. Diese wird hier jedoch mit **`switch () … case`** umgesetzt. Dies ist bei allen abzählbaren Datentypen möglich.

```
27  //-------------------------------
28     void loop() {
29          unsigned char out=0;
30          switch(PINC){
31          case 1: out=0x01; break;
32          case 2: out=0x02; break;
33          case 4: out=0x04; break;
34          case 8: out=0x08; break;
35          default: out=0xF0;
36          }
37          PORTD=out;
38     }
39  //-------------------------------
```

Abb. 12: Ausschnitt aus dem Quelltext bsp_07_switch_case

Entsprechend der digitalen Wertigkeit (1, 2, 4, 8. usw.) wird hier der Port C mit PINC überprüft. Im Falle einer Übereinstimmung wird die Auswahlstruktur direkt mit **`break`** verlassen.

Hier wird eine Zwischenvariable namens **`out`** benutzt, welche in Zeile 37 auf den Port D kopiert wird.

Aufgaben:

1 Überlegen Sie sich drei verschiedene Lichtmuster an Port D, die mit den Tasten an den Bits 3, 4 und 5 des Port C aufgerufen werden! Benutzen Sie hierbei **`if () .. else if () ..!`**

2 Setzen Sie nun Aufgabe 1 mit der Mehrfachauswahl **`switch () … case`** um. Benutzen Sie für die Auswahl nun die Bits 3, 4, und 5 des **Port B**. Die Ausgänge beleiben auf **Port D** !

8 bsp_08_count

Das Beispiel zeigt die Anwendung von Wiederholungsstrukturen (Schleifen) am Beispiel eines 8-Bit-Binärzählers.

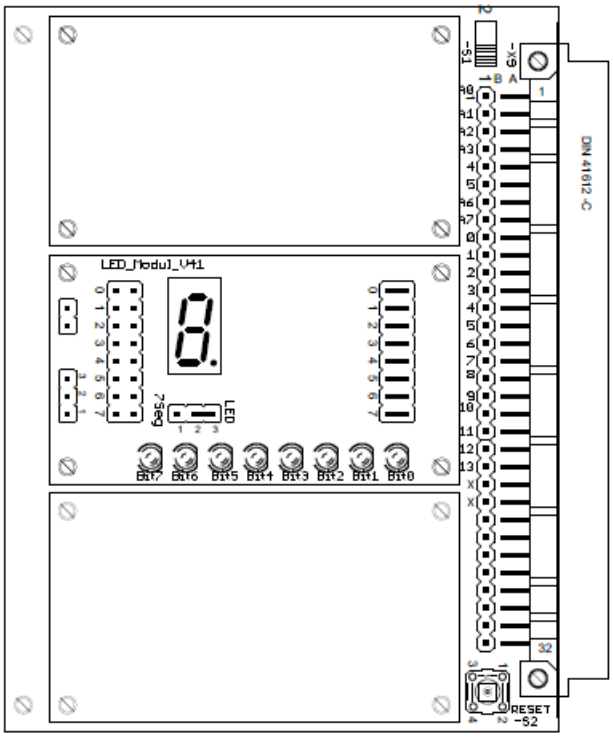

Abb. 13: Konfiguration der Module

Das Uno-Modul-Shield wird gemäß nebenstehender Abbildung konfiguriert (Jumper beachten).

Innerhalb der **`for ()-`** Schleife wird eine Variable bei jedem Durchlauf um 1 erhöht und auf den Port D geschrieben.

```
24   //----------------------------
25      void loop() {
26          unsigned char i;
27          for(i=0;i<=255;i++){
28              PORTD=i;
29              _delay_ms(100);
30          }
31      }
32   //----------------------------
```

Abb. 14: Ausschnitt aus dem Quelltext bsp_08_count

Aufgabe: Testen Sie auch die anderen beiden Schleifenvarianten mit **`while ()`** und **`do … while ()`** aus, indem Sie die entsprechenden Zeilen im Quelltext auskommentieren bzw. Kommentarzeichen entfernen.

9 bsp_09_lauflicht

Die Konfiguration des Uno-Modul-Shields ist identisch mit der von Beispiel 8. Hier werden innerhalb einer Schleife Bits um jeweils eine Stelle nach links geschoben (Zeile 29) und dann auf dem Port D optisch dargestellt.

```
23   //----------------------------
24      void loop() {
25      unsigned char i;
26      PORTD=0x01;
27          for(i=0;i<=7;i++){
28              _delay_ms(100);
29              PORTD=PORTD<<1;
30          }
31      }
32   //----------------------------
```

Abb. 15: Ausschnitt aus dem Quelltext bsp_09_lauflicht

Aufgaben:

1 Versuchen Sie ein Knight-Rider-Lauflicht zu erzeugen, welches periodisch hin und wieder zurück läuft.
2 Setzen Sie zusätzlich ein Switch-Modul ein und starten in Abhängigkeit davon, welche Taste betätigt wird, das Lauflicht verschieden schnell !
3 Implementieren Sie zwei up/down-Tasten, mit denen Sie einen Zähler bei Betätigung um eins inkrementieren bzw. dekrementieren !

10 bsp_10_bargraph

In diesem Beispiel soll ein Lauflicht über alle verfügbaren Portpins programmiert werden. Dazu müssen LED-Module auf alle Ports gesteckt werden (Abb. 16). Alternativ kann auch die Test_Unit (Abb. 17) über die 64polige Stiftleiste angesteckt werden.

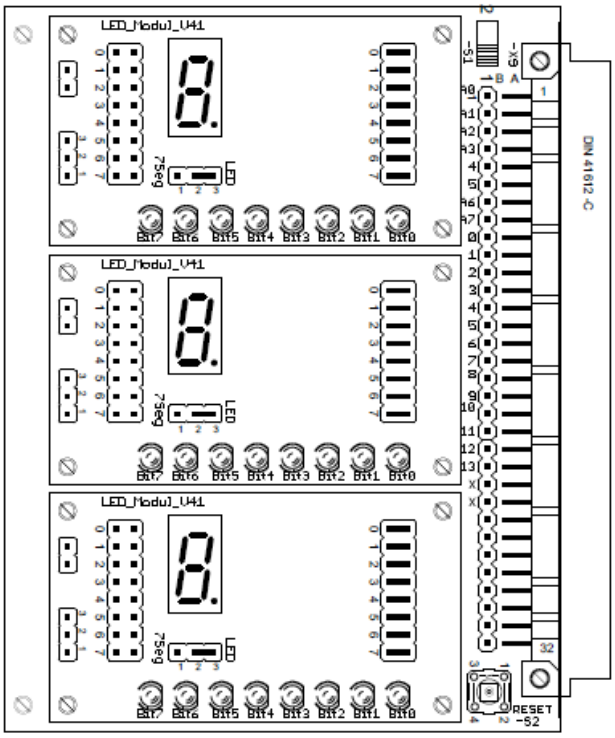

Abb. 16: Konfiguration der Module

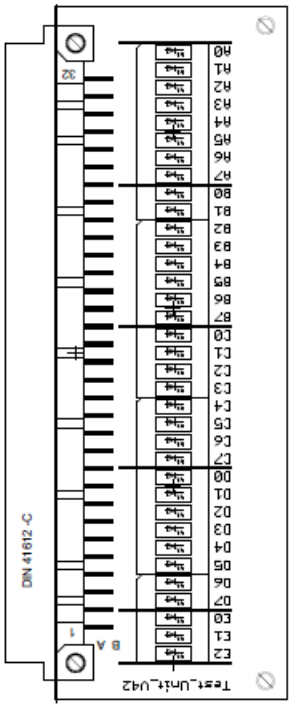

Abb. 17: Test-Unit für alle Portpins

```
25    //-----------------------------------------------------
26      void loop() {
27
28      PORTC=0x01;
29      for(i=0;i<=5;i++){_delay_ms(100);PORTC=PORTC<<1;}
30      PORTD=0x01;
31      for(i=0;i<=7;i++){_delay_ms(100);PORTD=PORTD<<1;}
32      PORTB=0x01;
33      for(i=0;i<=5;i++){_delay_ms(100);PORTB=PORTB<<1;}
34      }
35    //-----------------------------------------------------
```

Abb. 18: Ausschnitt aus dem Quelltext bsp_10_bargraph

Der Quelltext ist eine Erweiterung des Beispiels 9. Es wird stets das Bit 0 des betreffenden Ports auf 1 gesetzt, dann wird diese 1 mit jedem Schleifendurchlauf um 1 Stelle weiter geschoben und angezeigt.

Aufgabe: Ändern Sie den gegebenen Quelltext so ab, dass jeweils nur die geradzahligen Bits (0, 2, 4, 6) aufleuchten (ungeradzahligen Bits (1, 3, 5, 7) !

11 bsp_11_array

Im vorliegenden Beispiel soll dreimal von 0 bis 9 gezählt werden. Die Zahlen sollen auf der Siebensegment-Anzeige des LED-Moduls erscheinen (Jumperlage beachten, vgl. Abb. 19) !

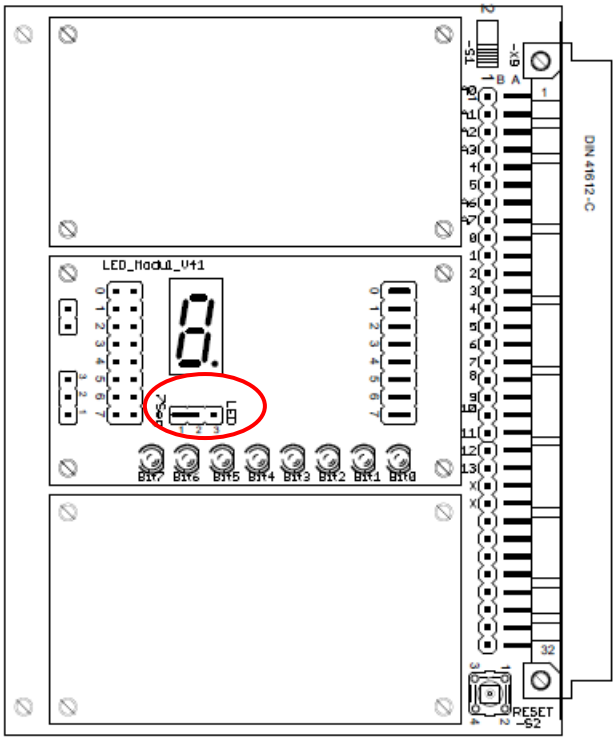

Abb. 19: Konfiguration der Module

Die Segmente der Anzeige entsprechen von Segment a bis zum Dezimalpunkt dp den Bits 0 bis 7 (Abb. 19a).

Die Bitkombinationen, die nötig sind, um die Zahlen von 0 bis 9 darzustellen, werden in dem array **_7segm[]** gespeichert, vgl. Zeile 26/27 in Abb. 20.

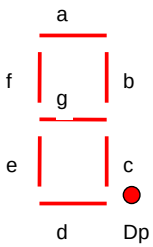

Abb. 19a: Segmente

Das Auslesen dieses arrays erfolgt in einer Schleife, indem der Index i jeweils um 1 erhöht wird, so ist z.B. **_7segm[0]= 0x3F**, dies ist genau die Bitkombination, um die Null darzustellen.

Wenn die Schleife (Zeile 28 bis 31) dreimal durchlaufen wurde, ist die globale variable **count =3**, so dass die Laufbedingung der Schleife in Zeile 28 nicht mehr gegeben ist.

```
24  //-------------------------------------------------------------------------
25      void loop() {
26         unsigned char i, _7segm[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,
27                                    0x7D,0x07,0x7F,0x6F};
28         while(count<3){      // count bei jedem Durchlauf um 1 erhöhen
29             for(i=0;i<=9;i++){ PORTD=_7segm[i];_delay_ms(300);}
30             count++;
31         }
32     }
33  //-------------------------------------------------------------------------
```

Abb. 20: Ausschnitt aus dem Quelltext bsp_11_array

Aufgabe: Ergänzen Sie den array **_7segm[]** mit den Buchstaben A, b, c, d, E und F, so dass auch einstellige Hexzahlen angezeigt werden können!

12 bsp_12_gray

In diesem Beispiel soll nicht im normalen 8421-Code hochgezählt werden, sondern im Gray-Code. Dies ist ein spezieller BCD-Code, welcher in der Übertragungstechnik oft benutzt wird, da sich zwischen zwei benachbarten Ziffern stets nur eine Bitposition verändert (vgl. Abb. 21). Die Konfiguration des Uno-Modul-Shields ist identisch mit der von Beispiel 11.

Nr.	1	2	3	4	5	6	7	8	9	10
Name	8-4-2-1-Code	Aiken-Code	unsymmetrischer 2-4-2-1-Code	Stibitz-Code (Exzeß-3-Code)	4-2-2-1-Code	White-Code	Gray-Code	Glixon-Code	O'Brien-Code II	reflektierter Exzeß-3-Code
Stellenwert / Dezimalziffer	8 4 2 1	2 4 2 1	2 4 2 1		4 2 2 1	5 2 1 1				
0	0000	0000	0000	0011	0000	0000	0000	0000	0001	0010
1	0001	0001	0001	0100	0001	0001	0001	0001	0011	0110
2	0010	0010	0010	0101	0010	0011	0011	0011	0010	0111
3	0011	0011	0011	0110	0011	0101	0010	0010	0110	0101
4	0100	0100	0100	0111	0110	0111	0110	0110	0100	0100
5	0101	1011	0101	1000	0111	1000	0111	0111	1100	1100
6	0110	1100	0110	1001	1010	1001	0101	0101	1110	1101
7	0111	1101	0111	1010	1011	1011	0100	0100	1010	1111
8	1000	1110	1110	1011	1110	1101	1100	1100	1011	1110
9	1001	1111	1111	1100	1111	1111	1101	1000	1001	1010

Abb. 21 aus: Grundlagen der Digitaltechnik, L. Borucki, 2. Auflage, S. 20, Teubner-Verlag

```
23    //-------------------------------------------------------------
24  void loop()
25  {
26     unsigned char i,
27     _7segm[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
28              0x7F,0x6F,0x77,0x7C,0x58,0x5E,0x79,0x71}, val=0x00,
29     gray[]={0x00,0x01,0x03,0x02,0x06,0x07,0x05,0x04,0x0C,0x0D};
30     for (i=0;i<=9;i++){
31         val=gray[i];
32         PORTD=_7segm[val];
33         _delay_ms(500);
34     }
35  }
36    //-------------------------------------------------------------
```

Abb. 21a: Ausschnitt aus dem Quelltext bsp_12_gray

Hier wird die entsprechende Zahl zunächst in dem array **`gray[]`** gespeichert. Über die Zwischenvariable **`val`** werden diese Werte dann zu den Indizes des arrays **`_7segm[].`** Dann erfolgt die periodische Ausgabe auf der Siebensegment-Anzeige.

Aufgabe: Ändern Sie den Quelltext so ab, dass wahlweise der Aiken- oder der Gray-Code durchlaufen wird. Zur Auswahl kann man das Switch-Modul mit den Tasten benutzen (Bit 0 betätigt→ Aiken-Code, Bit 1 betätigt→ Gray-Code, keine Taste betätigt → keine Codewandlung)!

13 bsp_13_taktmerker

In diesem Beispiel wird erstmals ein Interrupt verwendet, das ist eine Unterbrechung des Hauptzyklus, um die spezielle Interrupt-Funktion vorrangig abzuarbeiten. Zwischen Hauptzyklus und Interrupt-Funktion wird schnell hin- und hergeschaltet, so dass scheinbar zwei Aktionen parallel laufen. Die Konfiguration des Boards zeigt Abb. 22.

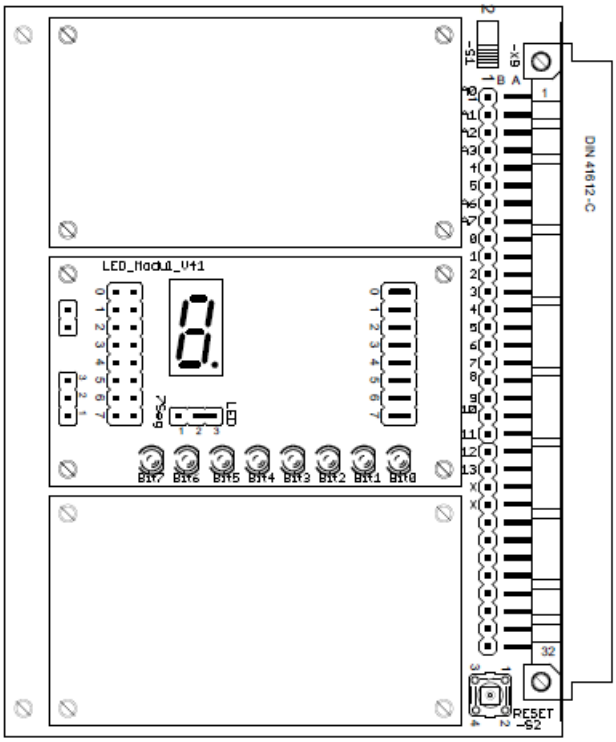

Abb. 22: Konfiguration der Module

Es gibt mehrere Ereignisse, die einen Interrupt auslösen können, in diesem Skript wird lediglich ein Timer-Überlauf benutzt. Dazu muss in der Funktion **`setup()`** die Zeit in µs definiert werden bis ein Aufruf der Interrupt-Funktion erfolgt, außerdem muss der Name der Interrupt-Funktion definiert werden (vgl. Zeilen 27, 28).

Die Interrupt-Funktion selbst sollte so kurz wie möglich gehalten werden, auf keinen Fall sollten in ihr Zeiten mit **`_delay_ms()`** implementiert werden. Im Beispiel wird in der Interrupt-Funktion namens **`timerISr`** das Bit 7 der Variable takt invertiert. Im Hauptzyklus wird auch auf diese Variabe zugegriffen, indem sie dem Port D zugewiesen wird. Daher muss diese Variable bei der Deklaration den Zusatz **`volatile`** erhalten (vgl. Zeile 18). Dadurch wird diese vom Compiler keinem Optimierungsprozess unterzogen und sie kann in anderen Funktionen benutzt werden. Die LED an Bit 7 blinkt schließlich mit der Periodendauer T = 200 ms.

```
16    #include <TimerOne.h>
17   //-------------------------------------------------------------------------
18    volatile unsigned char takt;
19
20    void setup() {
21    /* add setup code here, setup code runs once when the processor starts */
22    // DDRx: 1=output, 0=input
23      DDRC = 0xFF;         // Port C all outputs, 0-> input; 1-> output
24      DDRD = 0xFF;         // Port D all outputs
25      DDRB = 0xFF;         // Port B all outputs
26
27      Timer1.initialize(100000);          // set a timer of length 100.000µs=100ms
28      Timer1.attachInterrupt(timerIsr);   // attach the service routine here
29    }
30   //-------------------------------------------------------------------------
31    void loop() {
32    /* add main program code here, this code starts again each time it ends */
33      PORTD=takt;
34    }
35   //-------------------------------------------------------------------------
36    void timerIsr() {
37      takt=takt^0x80;      // Bit 7 blinkt
38    }
39   //-------------------------------------------------------------------------
```

Abb. 23: Auszug aus dem Quelltext von bsp_13_taktmerker

Für die Benutzung des Timer 1 ist es notwendig, eine spezielle Headerdatei einzubinden (vgl. Zeile 16). Diese muss in die Arduino-IDE eingebaut werden. Dazu befindet sich im Projektordner ein ZIP-komprimiertes Verzeichnis, mit dem das möglich ist.

Aufgaben:

1 Lassen Sie Bit 4 und 5 des **Port B** mit der oben beschriebenen Methode mit einer Frequenz von 1 Hz blinken!

2 Ergänzen Sie das Programm aus Aufgabe 1 folgendermaßen:
Es soll mit einer Taste an Pin 0 des **Port C** die LED an Bit 5 des Port D ein- und ausgeschaltet werden können! Die LEDs auf **Port B** sollen weiterhin blinken wie in Aufgabe 1 beschrieben!

14 bsp_14_LED_multiplex

In diesem Beispiel wird der Timer-Interrupt dazu benutzt, vier Siebensegment-Anzeigen nacheinander anzusteuern. Wenn dies schnell genug geschieht, ergibt sich ein stehendes Bild aller Anzeigen. Dies nennt man Multiplex-Betrieb. Man kann damit Ansteuerleitungen einsparen, indem man jeweils nur eine Anzeige freischaltet. Man benötigt dann 8 + 4 Leitungen für die Ansteuerung. Die 8 Leitungen für die einzelnen Segmente kommen von Port D, der genau 8 Bit hat, die weiteren 4 Leitungen zum Durchschalten werden über kurze Steckbrücken an die Bits 2, 3, 4 und 5 des Port C geschaltet (vgl. Abb. 24). Diese Konfiguration wird auch für die weiteren Beispiele beibehalten.

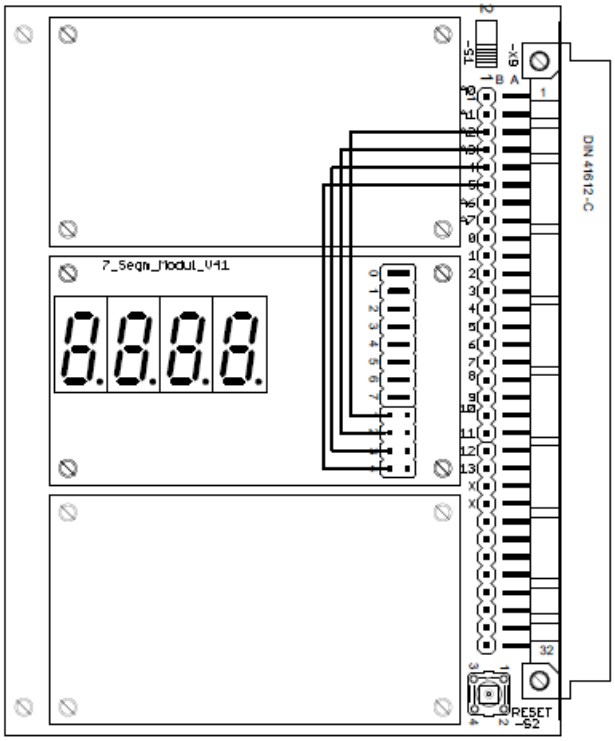

Abb. 24: Konfiguration der Module

In der Interrupt-Funktion werden die Anzeigen nacheinander durchgeschaltet. Im Hauptzyklus wird die Variable **`val`** an die Funktion **`data_to_7segm()`** übergeben (Zeile 37). Dort werden die Tausender, Hunderter, Zehner und Einer der Variablen auf die Anzeigen aufgeteilt. Letztlich erscheint auf der Anzeige der Wert der Variablen im Beispiel: 4321 (vgl. Zeile 36).

Das Umschalten der Anzeigen muss so schnell erfolgen, dass das menschliche Auge dem nicht mehr folgen kann. Dann läuft die Abarbeitung des Hauptzyklus scheinbar parallel zur Umschaltung der Siebensegment-Anzeigen.

```
33  //--------------------------------------------------------------------------
34   void loop() {
35   /* add main program code here, this code starts again each time it ends */
36     val=4321;               // Wert auf Siebensegment-Anzeige darstellen
37     data_to_7segm (val);
38     _delay_ms(100);
39   }
40
41  //--------------------------------------------------------------------------
42  // Interrupt-Service-Routine
43  //--------------------------------------------------------------------------
44     void timerIsr() {
45         static unsigned char mpx_count=0;
46         unsigned char ctrl_port, data_port,
47             zif[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F},
48             zifdp[]={0xBF,0x86,0xDB,0xCF,0xE6,0xED,0xFD,0x87,0xFF,0xEF};
49         PORTD=0x00;
50         ctrl_port=ctrl_port|0b00111100;
51         mpx_count++;
52         switch (mpx_count){
53             case 1: data_port=zif[digit_01];
54                     ctrl_port=ctrl_port&0b11111011; break;
55             case 2: data_port=zif[digit_02];
56                     ctrl_port=ctrl_port&0b11110111; break;
57             case 3: data_port=zif[digit_03];
58                     ctrl_port=ctrl_port&0b11101111; break;
59             case 4: data_port=zif[digit_04];
60                     ctrl_port=ctrl_port&0b11011111; mpx_count=0;
61         }
62         PORTC=ctrl_port;
63         PORTD=data_port;
64     }
65  //--------------------------------------------------------------------------
66     void data_to_7segm (unsigned int x) {
67         digit_01=x%10;              // Einer
68         digit_02=(x%100)/10;        // Zehner
69         digit_03=(x%1000)/100;      // Hunderter
70         digit_04=x/1000;            // Tausender
71     }
72  //--------------------------------------------------------------------------
```

Abb. 25: Ausschnitt aus dem Quelltext bsp_14_LED_multiplex

Aufgaben:

1 Verändern Sie die Zeit für das Leuchten einer Anzeige auf 1s und beobachten Sie den Umschaltvorgang!

2 Übertragen Sie verschiedene Werte auf die Anzeigen und testen Sie aus ab welcher Zeit die Anzeigen nicht mehr flackern !

15 bsp_15_stoppuhr

Das Beispiel zeigt eine Stoppuhr, die Konfiguration der Module ist aus Abb. 26 ersichtlich.

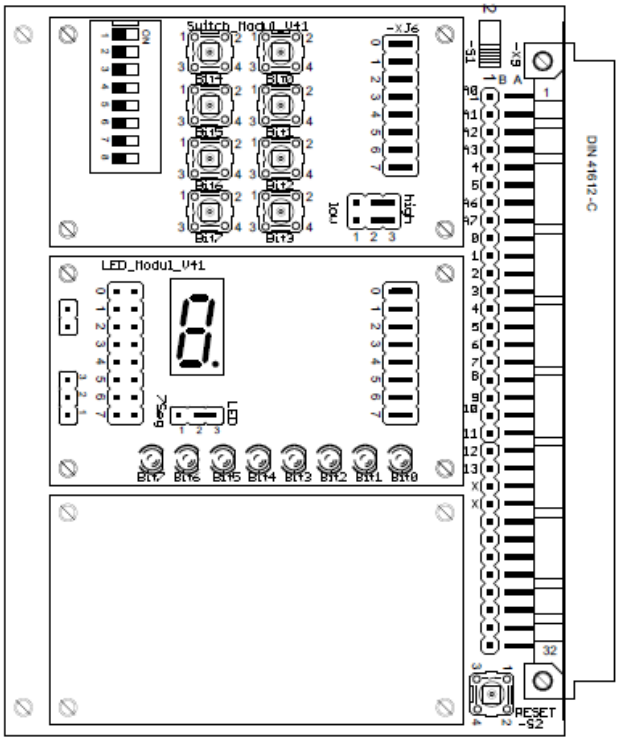

Abb. 26: Konfiguration der Module

Die Bedienung an Port C erfolgt folgendermaßen:

Bit 0 → Start
Bit 1 → Stopp
Bit 2 → Sekunden an Port D
Bit 4 → Minuten an Port D

In der Interrupt-Funktion werden nur die Hundertstelsekunden hochgezählt. Im Hauptzyklus erfolgt die Auswertung der Tasten und das Hochzählen der Sekunden und Minuten.
Wird keine Taste betätigt, so werden die Hundertstelsekunden auf Port D binär angezeigt.
Die Funktionen **`sei()`** und **`cli()`** erlauben bzw. löschen alle Interrupts (set bzw. clear all interrupts).

```
34  //-------------------------------------------------------------------------
35  void loop() {
36  /* add main program code here, this code starts again each time it ends */
37  unsigned char sek=0, minute=0;
38     while (1){
39
40          if(sek_01==100) { sek_01=0;sek++;}
41          if(sek==60) { sek=0;minute++;}
42
43          if (PINC& 0x01) {               // wenn start (Bit 0) betätigt
44              sei();
45              sek_01=0;
46              sek=0;
47              minute=0;
48          }
49          if (PINC& 0x02) cli();          // stop (Bit 1)
50          if (PINC& 0x04) PORTD=sek;          // Sekunden anzeigen (Bit 2)
51          else if (PINC& 0x08) PORTD=minute; // Minuten anzeigen (Bit 3)
52                  else PORTD=sek_01;          // Hundertstel anzeigen
53     }
54  }
55
56  //-------------------------------------------------------------------------
```

Abb. 27: Ausschnitt aus dem Quelltext bsp_15_stoppuhr

Aufgabe: Erweitern Sie die Stoppuhr um eine Stundenanzeige, diese soll mit der Taste an Bit 4 von Port C aktiviert werden.

16 bsp_16_LCD

Das LCD-Modul benötigt 6 Leitungen zum Mikrocontroller. Daher eignet sich der Port B mit 6 Bit besonders zum Anschluss des LCD-Moduls.

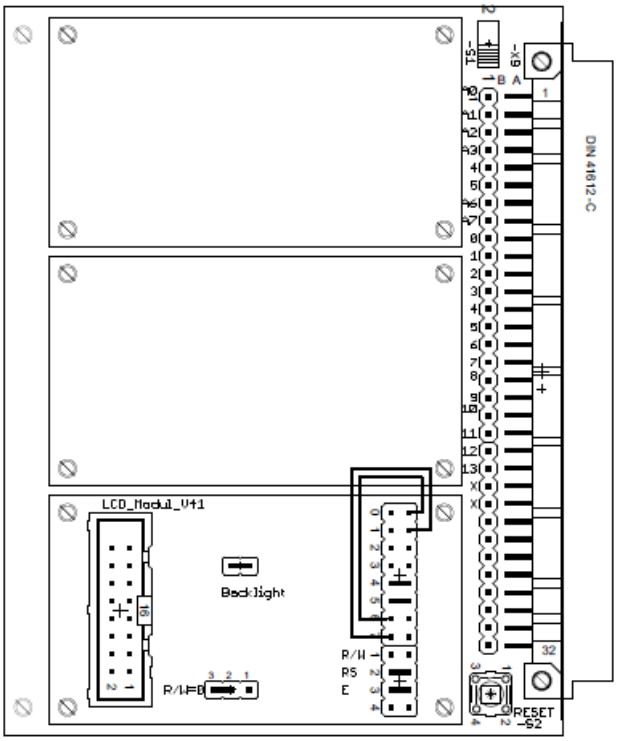

Abb. 28: Konfiguration der Module

Dabei müssen zwei Steckbrücken eingesetzt werden (vgl. Abb. 28).

In Zeile 19 werden die Arduino Pins den LCD-Pins zugeordnet.
In Zeile 29 erfolgt die Konfiguration des eingesetzten Displays:
20 Zeichen in 2 Zeilen
Dann wird in Zeile 0 mit 4 Leerstellen der Text ausgegeben (vgl. Zeile 30).
Um Variablen formatiert auszugeben, sollten diese vorher in Zeichen umgewandelt werden, dies geschieht bei Integer-Variablen mit der Funktion **`sprintf()`** und bei Floatvariablen mit **`dtostrf()`** (vgl. Zeilen 38, 41). Der array **`buf[]`** dient als Zwischenspeicher (Zeile 34).

```
19  LiquidCrystal lcd (10, 11, 12, 13, 8, 9);       // LCD-> Port B
20  // LiquidCrystal lcd(2, 3, 4, 5, 6, 7);         // LCD-> Port D
21  //-------------------------------------------------------------------------------------------
22 void setup(){
23     /* add setup code here, setup code runs once when the processor starts */
24     // PORTx: 1=output, 0=input
25     //DDRC = 0x00;       // Port C all inputs, 0-> input; 1-> output
26     //PORTC= 0x00;       // pull ups off, 0-> pull up off;  1-> pull up on
27     //DDRD = 0xFF;       // Port D all outputs
28     //DDRB = 0xFF;       // Port B all outputs
29     lcd.begin(20, 2);
30     lcd.setCursor(4, 0); lcd.print("Hallo Welt !");
31  }
32
33 void loop(){
34     char buf[10];
35     int val=1234;
36     float val_fl=3.1415;
37
38     sprintf(buf,"%4d ",val);                  // konvertiert int -> Zeichen
39     lcd.setCursor(2, 1); lcd.print(buf);      // nach 2 Zeichen beginnt Ausgabe,
40                                               // die 4 Zeichen benötigt
41     dtostrf(val_fl, 6, 4, buf);               // konvertiert double oder float -> Zeichen
42     //dtostrf(floatVar, minStringWidthIncDecimalPoint, numVarsAfterDecimal, charBuf);
43     lcd.setCursor(12, 1); lcd.print(buf);     // nach 12 Zeichen beginnt Ausgabe,
44                                               // die 6 Zeichen benötigt, 4 Nachkommastellen
45  }
46  //-------------------------------------------------------------------------------------------
```

Abb. 29: Auszug aus dem Quelltext von bsp_16_LCD

Aufgabe: Übertragen Sie verschiedene Wörter, ganze Zahlen (Typ Integer) und Kommazahlen (Datentyp float) an verschiedene Display-Positionen.

17 bsp_17_AD_Wandler

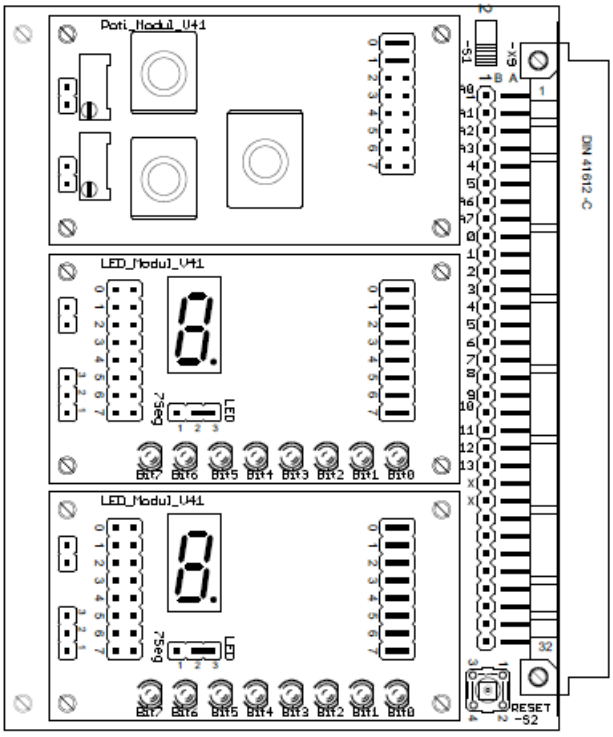

Abb. 30: Konfiguration der Module

Das Board sollte wie in Abb. 30 konfiguriert sein. Die beiden Potentiometer gelangen an die Bits 0 und 1 des Poti-Moduls.

Da ein 10 Bit-Wert binär dargestellt werden soll, benötigt man 2 LED-Module, auf Port D werden die unteren 8 Bit angezeigt, auf Port B gelangen die oberen 2 Bit.

Als Referenzspannung für den AD-Wandler wird die auf dem Uno-Modul-Shield bereitgestellte Referenz von 2,560 Volt benutzt. Damit ergibt sich eine Auflösung von 2,56V/1024= 2,5 mV. Dies wird in Zeile 30 vorbereitet.
Der Spannungswert wird mit der Spezialfunktion analogRead(0) in eine Zahl zwischen 0 und 1023 gewandelt.

```
20   //--------------------------------------------------------------------------------
21   unsigned int val=0;
22
23  //--------------------------------------------------------------------------------
24    void setup() {
25    /* add setup code here, setup code runs once when the processor starts */
26    // DDRx: 1=output, 0=input
27      DDRC = 0b11111100;   // Port C: 0-> input; 1-> output
28      DDRD = 0xFF;         // Port D all outputs
29      DDRB = 0xFF;         // Port B all outputs
30      analogReference(EXTERNAL);      // Referenzspannung auf 2,560 V abgleichen !
31    }
32
33  //--------------------------------------------------------------------------------
34    void loop() {
35    /* add main program code here, this code starts again each time it ends */
36      val=analogRead(0);       // Zahlen von 0 bis 1023 -> val
37      PORTD= val&0xFF;         // 8 lower bits -> Port D
38      PORTB=(val&0x300)>>8;    // 2 upper bits -> Port B
39    }
40  //--------------------------------------------------------------------------------
```

Abb. 31: Auszug aus dem Quelltext von bsp_17_AD_Wandler

Aufgaben:

1 Ändern Sie den obigen Quelltext so ab, dass die Spannung des zweiten Potentiometers auf dem Modul gewandelt wird!

2 Was geschieht, wenn die Eingangsspannung des Wandlers größer als 2,56 V wird? Messen Sie nach!

18 bsp_18_LCD_Voltmeter

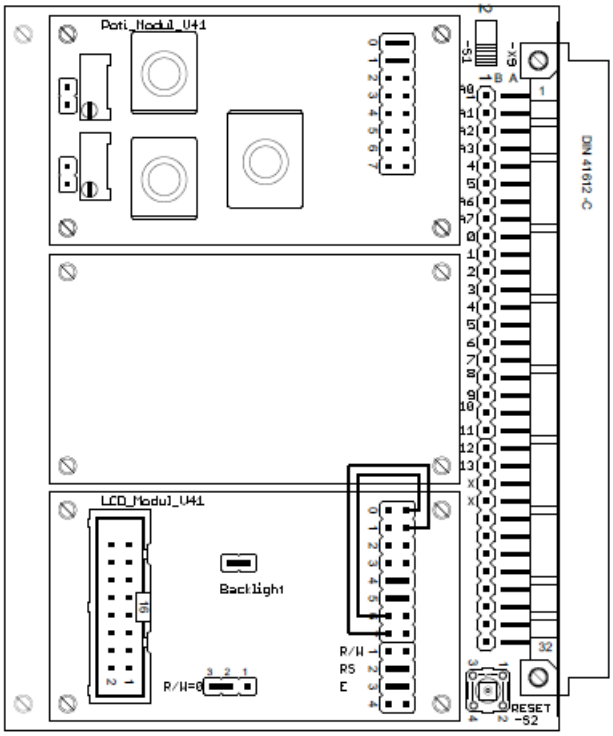

Abb. 32: Konfiguration der Module

In diesem Beispiel wird die Spannung des ersten Potentiometers auf dem LC-Display dargestellt. Das Board sollte wie in Abb. 32 konfiguriert sein.

Zuerst wird der Analogwert in eine ganze Zahl zwischen 0 und 1023 gewandelt (Zeile 42). Dann erfolgt in Zeile 43 die Eichung auf den gesamten Bereich, nämlich 2,56 Volt. Nun liegt eine Kommazahl vor, die in Zeile 55 in eine Zeichenkette gewandelt wird. Dabei sollen 5 Zeichen entstehen, inclusive dem Kommazeichen bei 3 Nachkommastellen. In Zeile 46 wird die Zeichenkette dann auf das Display übertragen.

```
36 //-------------------------------------------------------------------------------------------
37     void loop() {
38         unsigned int val=0;
39         float sp=0;
40         char buf[10];
41 
42         val= analogRead(0);              // Wert aus dem AD-Wandler
43         sp=(val*2.560)/1024;             // Spannung in Volt
44         dtostrf(sp, 5, 3, buf);          // float -> string
45     //  dtostrf(floatVar, minStringWidthIncDecimalPoint, numVarsAfterDecimal, charBuf);
46         lcd.setCursor(8, 1); lcd.print(buf);
47         lcd.setCursor(14, 1); lcd.print("V");
48 
49     }
50 
51 //-------------------------------------------------------------------------------------------
```

Abb. 33: Auszug aus dem Quelltext von bsp_18_LCD_Voltmeter

Aufgaben:

1 Stellen Sie beide Potentiometer-Spannungen auf dem LC-Display dar!

2 Bauen Sie auf Port D ein LED-Modul ein und stellen die Spannung eines Potentiometers zusätzlich als Bargraphanzeige bis 2,55 V dar. Die Abschnitte sollten dabei gleichverteilt sein!

19 bsp_19_LED_Voltmeter

Die Spannung am ersten Potentiometer soll auf der Siebensegment-Anzeige in der Einheit Volt angezeigt werden, außerdem soll an Port D eine Analoganzeige in Form eines Balkens realisiert werden.

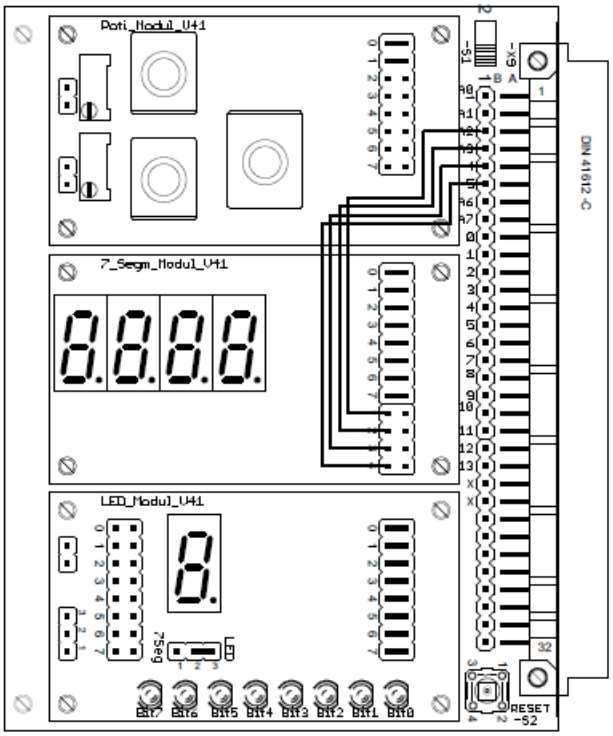

Abb. 34: Konfiguration der Module

Zunächst wird der Mittelwert aus 10 Messungen gebildet (Zeilen 41, 42), dann wird die variable **`val`** mit dem Wert der Referenzspannung (Full-Scale-Wert) geeicht und der Funktion **`data_to_7segm ()`** übergeben.
Die analoge Bargraphanzeige mit 6 Bit wird mit einer Auswahlstruktur realisiert (Zeilen 45 bis 51).

Für die Funktion **`round()`** wird die Headerdatei **`math.h`** benötigt, diese ist mit einzubinden mit:
`# include <math.h>`

```
37  //-------------------------------------------------------------------------------
38    void loop() {
39    /* add main program code here, this code starts again each time it ends */
40      val=0;
41      for(int j=0;j<=9;j++) val =val+analogRead(0);
42      val=val/10;                                  // Mittelwert aus 10 Messungen
43      u= int(round(val*2.56/1.024));
44      data_to_7segm (u);                           // Ausgabe auf die Siebensegmentanzeige
45      if (val>=855) PORTB=0b00111111;              // zusätzliche Leuchtbandanzeige
46      else if (val>=685) PORTB=0b00011111;
47      else if (val>=515) PORTB=0b00001111;
48      else if (val>=345) PORTB=0b00000111;
49      else if (val>=175) PORTB=0b00000011;
50      else  if  (val>=1) PORTB=0b00000001;
51      else PORTB=0x00;
52      _delay_ms(100);
53    }
54
55  //-------------------------------------------------------------------------------
```

Abb. 34a: Auszug aus dem Quelltext von bsp_19_LED_Voltmeter

Aufgabe: Bauen Sie auf **Port B** ein Switch–Modul ein, mit dem die Spannungen, die auf die Siebensegment-Anzeigen gelangen, ausgewählt werden können. Ist der Schiebeschalter 1 betätigt, soll die Spannung des ersten Potis angezeigt werden, bei 2 die Spannung des zweiten Potis. Ist kein Kanal angewählt, soll das Display **0000** anzeigen.

20 bsp_20_Encoder

Das vorliegende Beispiel soll die Verwendung eines Drehencoders verdeutlichen. Das Board ist gemäß Abb. 35 zu konfigurieren (Steckbrücke beachten).

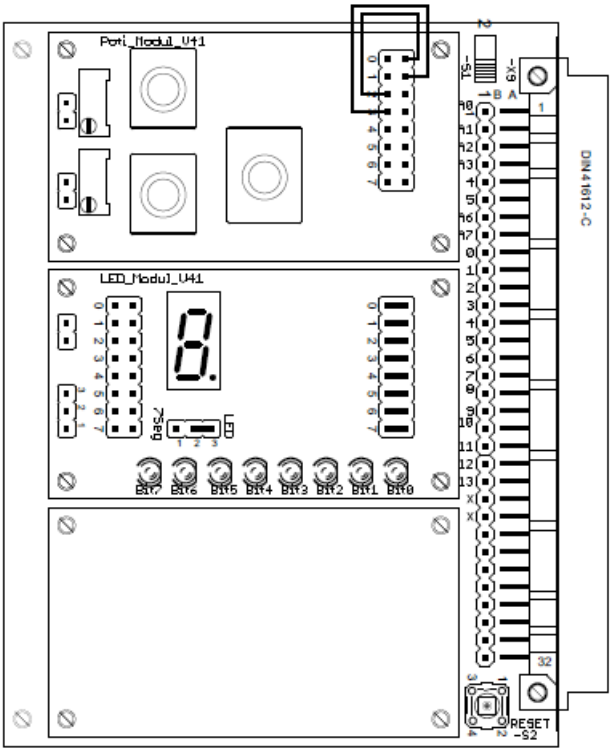

Abb. 35: Konfiguration der Module

Drehimpulsgeber (Encoder) kann man als „digitale Potentiometer“ bezeichnen. Sie erzeugen beim Verstellen zwei zeitlich versetzte Signale an den beiden Kanälen A und B (Abb. 36, 37). Hiermit wird die Drehrichtung bestimmt. Im Gegensatz zum normalen Potentiometer besitzen sie keine Endanschläge. Meist haben sie eine Rastung, um stabile Schaltpunkte zu erhalten. Oftmals ist am Ende der Achse ein normaler Tastschalter (okay-Taster) integriert.

Abb. 36: Encoder

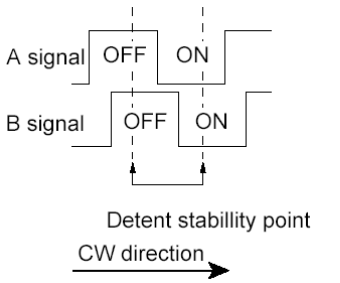

Abb. 37: Impulse ALPS STEC 11

Die beiden Kanäle werden mit Steckbrücken auf die Bits 0 und 1 des Port C geschaltet (Zeilen 17, 18).

```
17      #define Channel_A     PINC&0x01         // an Pin Bit 0
18      #define Channel_B     PINC&0x02         // an Pin Bit 1
19      #define LEDs          PORTD             // Ausgabe Port D
```

Abb. 38: Auszug aus dem Quelltext bsp_20_Encoder

Die am Markt erhältlichen Encoder unterscheiden sich in der Anzahl der erzeugten Impulse pro Umdrehung, der Lage der stabilen Schaltpunkte sowie der Anzahl der Rastungen. Im Folgenden wird beispielhaft eine Funktion namens **`encoder_read()`** erstellt, um Encoder der Baureihe ALPS STEC 11 auszulesen (Zeilen 43 bis 50). Diese Funktion registriert Änderungen auf den beiden Kanälen und liest aus dem array **`tab[]`** die entsprechenden Werte aus, die dann im Hauptzyklus weiterverarbeitet werden. Die Abtastung des Encoders erfolgt zyklisch innerhalb eines Timer-Interrupts.

```
35  //-------------------------------------------------------------------------------
36      void loop() {
37      /* add main program code here, this code starts again each time it ends */
38          val+=encoder_read();      // zählt bei jeder Raste um 1 hoch bzw. runter
39          LEDs=val;
40      }
41
42  //-------------------------------------------------------------------------------
43      char encoder_read (void) {      // Encoder auslesen
44          char value;
45          cli();                      // disable interrupts
46          value=enc_delta;
47          enc_delta=0;
48          sei();                      // enable interrupts
49          return value;
50      }
51
52  //-------------------------------------------------------------------------------
53  // Interrupt-Service-Routine
54  //-------------------------------------------------------------------------------
55      void timerIsr() {
56          static unsigned char last=0;     // Wert speichern auf fester Adresse
57                                           // dabei einmal initialisieren
58          char tab[]= {0,0,-1,0,0,0,0,1,1,0,0,0,0,-1,0,0};
59              // Indizes 7 und 8 zählen hoch, Indizes 2 und 13 runter
60              // für volle Auflösung: {0,1,-1,0,-1,0,0,1,1,0,0,-1,0,-1,1,0}
61          last=(last<<2)&0x0F;             // 2 nach links und maskieren
62          if(Channel_A)last|=1;            // Änderung auf Kanal A
63          if(Channel_B)last|=2;            // Änderung auf Kanal B
64          enc_delta+=tab[last];            // Anderung Encoderwert 0, 1 oder -1
65      }
66
67  //-------------------------------------------------------------------------------
```

Abb. 39: Auszug aus dem Quelltext von bsp_19_LED_Voltmeter

Aufgabe: Ändern Sie den Quelltext so ab, dass bei jedem Klick in Zehnerschritten hoch- bzw. runtergezählt wird.

21 bsp_21_Encoder_LCD:

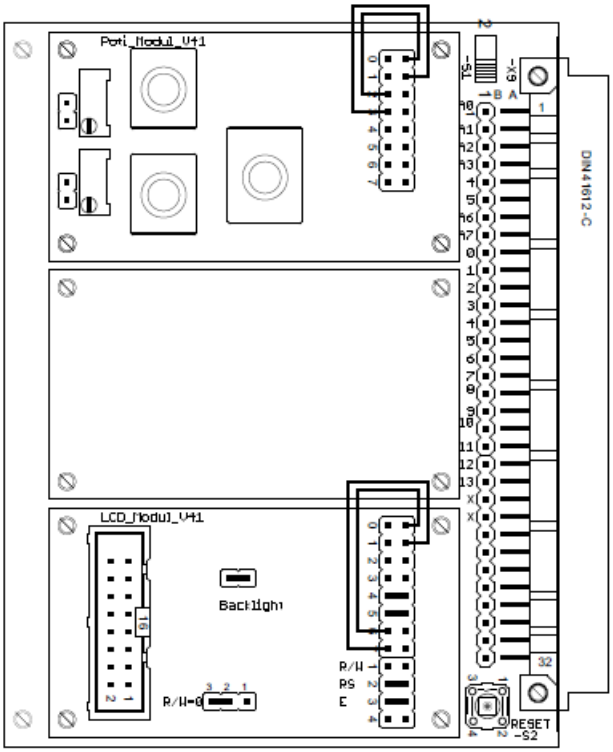

Abb. 40: Konfiguration der Module

Im Beispiel soll der Encoder Zahlen im Bereich von – 30 … + 30 liefern und auf dem LC-Display anzeigen.

Dazu wird der Encoder zyklisch ausgelesen und die Werte über den Umweg der Konvertierung in eine Zeichenkette an die LCD gesendet.

Die Abtastung des Encoders erfolgt wie im letzten Beispiel innerhalb eines Timer-Interrupts.

```
42  //-----------------------------------------------------------------------
43  void loop() {
44    /* add main program code here, this code starts again each time it ends */
45      char buf[10];
46      static int val=0;        // initialisiert beim ersten Mal mit 0
47      val+=encoder_read();     // zählt bei jeder Raste um 1 hoch bzw. runter
48      if(val<-30) val=-30;
49      if(val>30) val=30;
50      sprintf(buf,"%4d ",val);                // konvertiert int -> Zeichen
51      lcd.setCursor(12, 0); lcd.print(buf);   // nach 12 Zeichen beginnt
52                                              // vierstellige Variable
53  }
54
55  //-----------------------------------------------------------------------
```

Abb. 41: Auszug aus dem Quelltext von bsp_21_Encoder_LCD

Aufgaben:

1 Ändern Sie den Quelltext so ab, dass die Zahlen zwischen -50 und +70 dargestellt werden!

2 Nun soll der Taster des Encoders dazu benutzt werden, um eine gröbere Auflösung von 10 pro Klick einzustellen! (Tip: Unterlagen zum Poti-Modul beachten!)
Bei nochmaligem Tasten soll wieder 1 pro Klick gelten !

3 Zusätzlich zur LCD-Anzeige sollen die Zahlen in binärer Darstellung an Port D angezeigt werden! Führen Sie die notwendigen Hard- und Softwareänderungen durch !

22 bsp_22_Encoder_7segm:

Nun soll die Zahl, die mit dem Encoder eingestellt wird, auf der Siebensegment-Anzeige (Port D) sowie auf der LED-Anzeige (Port B) erscheinen im Bereich 0 ... 30.

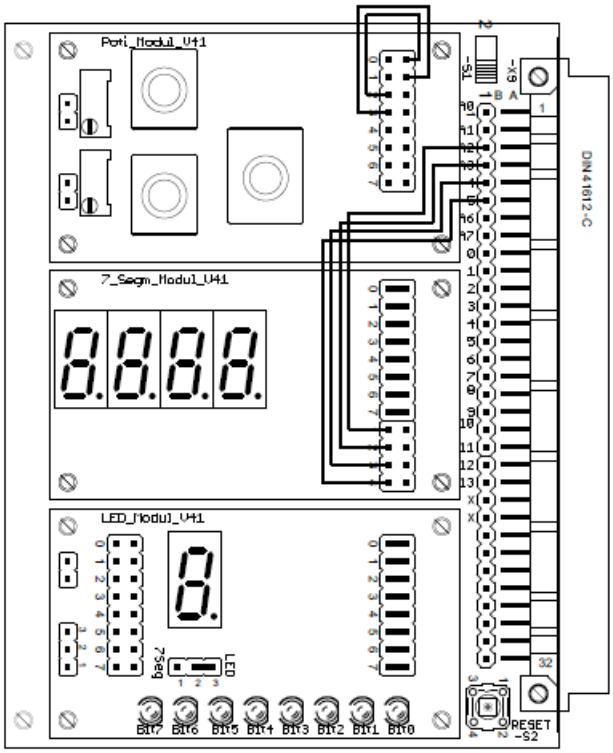

Abb. 42: Konfiguration der Module

Dazu wird der Encoder zyklisch ausgelesen, die Werte werden über die Funktion **`data_to_7segm()`** an die Anzeigen gesendet (Zeile 46).

Ferner werden die Werte auch dem Port B zugewiesen (Zeile 47).

Die Abtastung des Encoders erfolgt wie im letzten Beispiel innerhalb eines Timer-Interrupts.

```
40  //--------------------------------------------------------------------------------
41   void loop() {
42   /* add main program code here, this code starts again each time it ends */
43     val+=encoder_read();       // zählt bei jeder Raste um 1 hoch bzw. runter
44     if(val<0) val=0;
45     if(val>30) val=30;
46     data_to_7segm (val);
47     PORTB=val;
48
49   }
50
51  //--------------------------------------------------------------------------------
```

Abb. 43: Auszug aus dem Quelltext von bsp_22_Encoder_7segm

Aufgaben:

1 Der Encoder soll in Hunderterschritten hoch zählen zwischen 0 und 2000!

2 Bauen Sie auf Port B ein Switch-Modul ein, betätigt man die Taste an Bit 0 soll in Zehnerschritten hochgezählt werden, betätigt man die gleiche Taste noch einmal, soll der Encoder wieder Einerschritte liefern im Bereich 0 bis 100!