Aufgabenblatt 5 Termine: KW 20, KW 21, KW 22

Gruppe	
Name(n)	Matrikelnummer(n)

Flüssigkristalldisplays (LCDs)sind ein häufig verwendetes Ausgabegerät für eingebettete Systeme. Dem Bedarf entsprechend kann auf einfache Zeichen-Displays oder auf grafische Displays zurückgegriffen werden. Für die Aufgaben dieses Blatts werden Sie ein grafisches 1.8" TFT-LCD (Thin-film-transistor liquid-crystal display) einsetzen, um damit die grundlegende Funktionalität zur visuellen Ausgabe zu implementieren.

Die Kommunikation mit dem Display findet über einen **SPI**-Bus statt. SPI steht für Serial Peripheral Interface und stellt einen Standard für einen synchronen seriellen Datenbus dar.

- **SPI**: **S**erial **P**eripheral **I**nterface
  Serieller Bus, **Master/Slave** Kommunikation, synchroner Datentransfer

Die Schnittstelle zum eigentlichen Display wird durch Displaycontroller (Sitronix ST 7735R) bereitgestellt. Setzen Sie sich mit dem Datenblatt des Displaycontrollers auseinander. Der Displaycontroller verfügt über eine parallele, eine serielle 3-Draht und eine serielle 4-Draht Schnittstelle. Über die äußere Beschaltung ist die serielle 4-Draht Schnittstelle fest konfiguriert. Abb. 1 zeigt das vereinfachte Timing-Diagramm. Im Gegensatz zum SPI-Bus ist hier die Datenleitung (SDA) bidirektional. Ist eine Ansteuerung des Displays über den o. g. SPI-Bus des Arduino DUE überhaupt realisierbar?

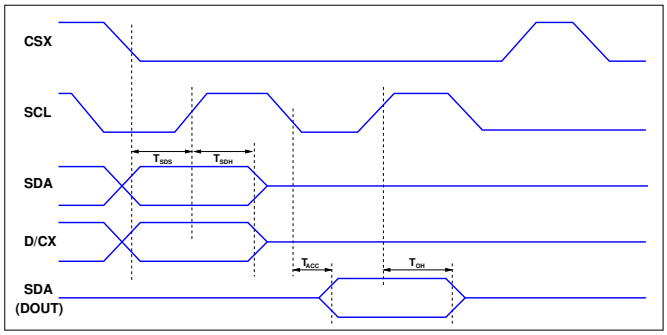

Abbildung 1: Vereinfachtes Timing-Diagramm der 4-Draht Schnittstelle des ST7735R (vgl. Datenblatt ST7735R, S.25)

Aufgabe 5.1

Diskutieren Sie die in Abb. 2 vorgeschlagene Beschaltung. Welche Bedeutung hat der Widerstand R_S? Könnte er auch fortgelassen werden? Was können Sie über die Dimensionierung aussagen?

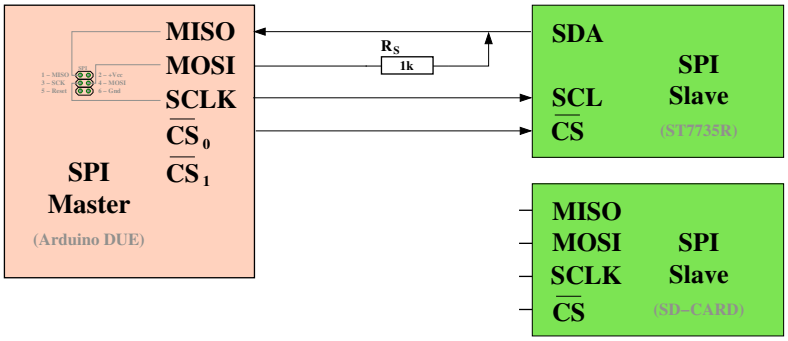

Abbildung 2: Anschlussvorschlag des Displays mit St7735R-Controller

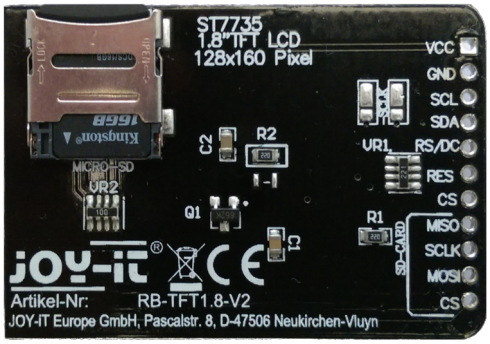

Abbildung 3: Rückseite des Dispalys mit SD-Card-Slot.

Verbinden Sie das Display mit dem Arduino DUE und testen Sie Ihren Aubau mit dem auf der Website der Übung bereitgestellten Testprogramm. Für die Lösung der Aufgaben wird empfohlen, den Versuchsaufbau gemäß Abbildung 4 zu verdrahten. Beachten Sie bei der Verdrahtung folgende Hinweise:

- Verbinden Sie **VCC** mit **3.3 V** und stellen Sie zusätzlich die Masseverbindung **GND** her.
- Verbinden Sie **SCL** des Displays mit **SCK** (SPI Clock) und **SDA** des Displays über den Serienwiderstand von 1 kΩ mit **MOSI** (SPI Master Out Slave In) des SPI-Headers des Arduino DUE (vgl. Arduino Due Pinbelegung).
- **MISO** (SPI Master In Slave Out) wird für die Display-Ansteuerung (s.o.) nicht zwingend benötigt.
- Verwenden Sie für die Verbindung der Anschlüsse **RS/DC** (Data/Command des Displays) und **RES** (Hardware-Reset des Displays, active-low) beliebige digitale Anschlusspins des Arduino Due. Im Beispieldesign wird für RS/DC Pin 8 und für RES Pin 9 verwendet.

- Verbinden Sie **TFT_CS** mit einem der freien Hardware-gesteuerten `Slave-Select` Anschlusspins des Arduino Due: **10** oder **52**. In Abb. 4 wird Pin **10** verwendet.

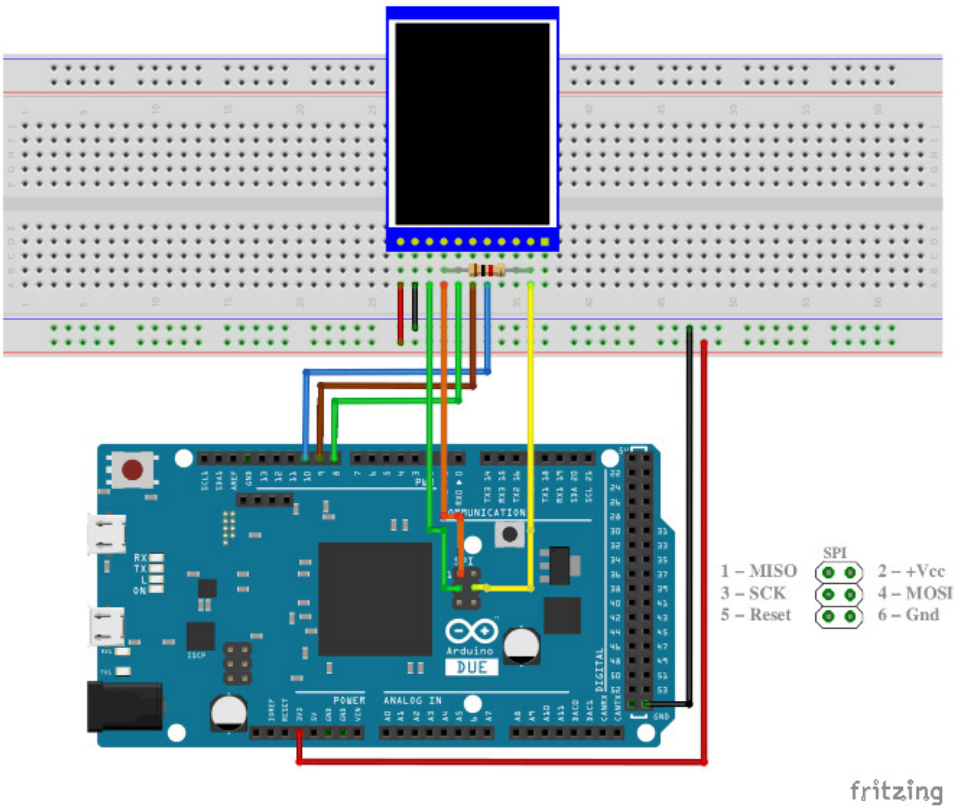

Abbildung 4: Vorschlag für die Verdrahtung des Versuchsaufbaus.

Aufgabe 5.2

Entwickeln Sie ein Programm, das folgende Funktionalität zur Verfügung stellt:

1. Initialisierung des Displays bei Inbetriebnahme des Systems: `setup()`.
2. Pixel-weise Manipulation des Displayinhalts in einem Bildspeicher („Frame-Buffer“): z.B. `setPixel(...)`.
3. Demo-Funktion, die ein wiederkehrendes Muster auf dem Display darstellt.

Das Arduino Framework macht Ihnen die Verwendung der seriellen SPI Schnittstellen einfach. Betrachten Sie die Dokumentation der Bibliothek `SPI` und verschaffen Sie sich einen Überblick über den Ihnen zur Verfügung stehenden Umfang an Funktionen. Vergessen Sie nicht die `SPI` Bibliothek explizit in den Quellcode einzubinden (`#include <SPI.h>`).

Hinweis: Beim Arduino Due besitzen die Funktionen `SPI.begin`, `SPI.beginTransaction` und

`SPI.transfer` im Gegensatz zu den üblichen AVR-Boards noch einen weiteren Parameter[1], den *Slave-Select* bzw. *Chip-Select* Anschlusspin.

Im Folgenden sind die Funktionen für den DUE aufgelistet, die zur Bearbeitung der Aufgabe benötigt werden:

```
* SPI.begin(⟨sce-pin⟩)                                    → SPI.begin
* SPI.beginTransaction(⟨sce-pin⟩, ⟨settings⟩)              → SPI.beginTransaction
* SPI.endTransaction()                                    → SPI.endTransaction
* SPI.transfer(⟨sce-pin⟩, ⟨data⟩)                         → SPI.Transfer
```

Hinweise zu Punkt 1:

- Zunächst ist es notwendig ein Reset des Displays durchzuführen. Beachten Sie bitte, dass die `RST` Leitung **low-active** ist, d.h. der Reset erfolgt nur dann, wenn der Signalpegel auf `LOW` gesetzt wird. **Wichtig:** Für das Display ist eine mindestens 10us lange `RESET`-Phase erforderlich.
- Initialisieren Sie einen Datenübertragungsvorgang wie folgt:

  ```
  SPI.beginTransaction(⟨sce-pin⟩, SPISettings(...))
  ```

  `SPISettings` stellt eine Datenstruktur mit für die Übertragung relevanten Parametern dar, die Sie mit folgendem Konstruktor direkt initialisieren können:

  ```
  SPISettings(⟨max-speed⟩, ⟨data-order⟩, ⟨data-mode⟩)
  ```

  Geben Sie für ⟨*max-speed*⟩ zunächst **1 MHz** in Hertz an. Der Displaycontroller erwartet das MSB (most significant bit) zuerst, geben Sie also die entsprechende Konstante für ⟨*data-order*⟩ an. ⟨*data-mode*⟩ spezifiziert die für die Datenübertragung relevanten Zustände des Taktsignals, verwenden Sie hier die entsprechende Konstante für `MODE_0` (d.h. Bit-Übertragung bei **fallender Flanke**, Inaktivität des Taktsignals bei Signalpegel `LOW`).
- Übertragen Sie die Daten Byte-weise an das LC-Display:

  ```
  SPI.transfer(⟨sce-pin⟩, ⟨data⟩)
  ```

- Terminieren Sie den Datenübertragungsvorgang wie folgt:

  ```
  SPI.endTransaction()
  ```

- Beachten Sie bitte, dass das Display zwischen **LCD-COMMAND** (`D/C = LOW`) und **LCD-DATA** (`D/C = HIGH`) unterscheidet. Bevor Sie das Display tatsächlich benutzen können, sind die folgenden Befehle (**LCD-COMMAND**) als Initialisierungssequenz an das Display zu übertragen. Dabei sind abhängig vom jeweiligen Befehl noch die erforderlichen Parameter - dann als Daten (**LCD-DATA**) - zu übertragen. Die angegebene Initialisierungssequenz initialisiert nur die Register des Displaycontrollers, die entweder nach HW- bzw. SW-RESET gar nicht initialisert wurden, oder die abweichend von geladenen Default-Werten gesetzt werden sollten. Ferner ist zu beachten, dass die Befehle SWRESET und SLPOUT ca. 120 mS für die Ausführung benötigen (siehe dazu das Datenblatt, ab S. 79):

  1. `0x01` : SWRESET (software reset)

[1] Anders als in der SPI-Dokumentation beschrieben.

2. `0x11` : SLPOUT (Sleep out & booster on)
3. `0x36` : MADCTL (Memory Data Access Control)
   `0x08` (Row-, Column-, Exchange-, Refresh-, RGB-, Refresh-Order)
4. `0x3A` : COLMOD (GB-format, 12/16/18bit
   `0x55` (16-bit/pixel)
5. `0x2A` : CASET (Column adress set)
   `0x00` (first column, high order byte)
   `0x02` (first column, low order byte)
   `0x00` (last column, high order byte)
   `0x81` (last column, low order byte)
6. `0x2B` : RASET (Row adress set
   `0x00` (first row, high order byte)
   `0x01` (first row, low order byte)
   `0x00` (last row, high order byte)
   `0x80` (last row, low order byte)
7. `0x13` : NORON (Partial off (Normal))
8. `0x29` : DISPON(Display on)

Hinweise zu Punkt 2:

- Mit dem Befehl COLMOD der obigen Initialisierungssequenz wird die Wortbreite für einen RGB-Farbwert auf 16 bit festgelegt und das Display erwartet die Farbwerte im RGB 565 Format. Machen Sie sich mit dieser Kodierung vertraut.
- Das Display besitzt eine Auflösung von 128×160 Pixeln (Zeilen $\times$ Spalten). Eine Adressierung einzelner Pixel wird durch den Befehlssatz des Displaycontrollers (RAMWR) unterstützt. Bevor in den Displayspeicher des Controllers geschrieben werden kann, muss mittels der entsprechender Befehle ein Schreib-Fenster über den Spalten- und Zeilenbereich definiert werden (siehe Datenblatt, S. 61 ff).
- Um den Kommunikationsaufwand mit dem Display gering zu halten und größere grafische Operationen schneller erledigen zu können, soll ein Pufferspeicher für den Displayinhalt definiert werden. Im Pufferspeicher finden alle „Zeichenoperationen“ statt. Nach Abschluss aller Operationen wird der Pufferspeicher komplett an das Display übertragen.
- Um grafische Operationen später möglichst flexibel umsetzen zu können, soll der Pufferspeicher Pixel-weise adressierbar sein. Implementieren Sie also eine Funktion die an der $\langle x \rangle, \langle y \rangle$-Position das Pixel den gewünschten 16-bit Farbwert setzt: `setPixel(int x, int y, uint16_t value)`.

Hinweise zu Punkt 3:

- Zur Demonstration der Korrektheit der entwickelten Funktionalität soll folgende Funktion implementiert werden:
  1. Beginnen Sie mit der ersten Spalte des Displays und **aktivieren** Sie jeden Pixel dieser Spalte.
  2. Aktualisieren Sie die Darstellung des Displays.
  3. Warten Sie 20ms ab, fahren Sie mit der nächsten Spalte des Displays fort und aktivieren Sie auch hier jeden Pixel.
  4. Wiederholen Sie die Schritte 1-3, bis Sie jede Spalte aktiviert haben.
  5. Beginnen Sie erneut mit der ersten Spalte des Displays und **deaktivieren** Sie jeden Pixel dieser Spalte.
  6. Aktualisieren Sie die Darstellung des Displays.
  7. Warten Sie 20ms ab, fahren Sie mit der nächsten Spalte des Displays fort und deaktivieren Sie auch hier jeden Pixel.
  8. Wiederholen Sie die Schritte 5-7, bis Sie jede Spalte deaktiviert haben.
  9. Fahren Sie mit Schritt 1. fort.

Aufgabe 5.3

Verwenden Sie den auf der Webseite zur Verfügung gestellten ASCII-Datensatz und implementieren Sie mithilfe der existierenden Funktionalität aus der vorherigen Aufgabe folgende Funktion:

- `printChar(int x, int y, char value, uint16_t fgColor, unit16_t bgColor)`

Die Funktion soll es Ihnen ermöglichen ein im Datensatz codiertes Zeichen an einer beliebigen Stelle des Displays zu positionieren. **ACHTUNG:** Vergessen Sie dabei nicht die Grenzen des Displaypuffers zu prüfen! Ihre Funktion soll also einen Rückgabewert liefern (z.B. -1) falls die Angabe der Koordinaten es nicht ermöglicht das Zeichen vollständig darzustellen.

Entwickeln Sie zur einfachen Ausgabe von Strings folgende zusätzliche Funktion:

- `printString(int x, int y, char *c_str,  uint16_t fgColor, unit16_t bgColor)`

Die Funktion soll keine Zeilenumbrüche produzieren. Lässt sich ein String nicht vollständig in einer Zeile darstellen, so soll die Funktion dieses mit einem entsprechenden Rückgabewert quittieren.

Erstellen Sie zur Demonstration Ihrer bisherigen Arbeit eine Funktion mit dem Namen `runStudentIdDemo()`, die abwechselnd die Kombination aus Vorname/Nachname und der Matrikelnummer für jeden Teilnehmer Ihrer Gruppe auf dem Display darstellt. Der Wechsel zwischen den Datensätzen soll alle 5 Sekunden geschehen. Positionieren Sie die Matrikelnummer in einer neuen Zeile. Sollte Ihr Name zu lang für eine Displayzeile sein können Sie beliebig abkürzen oder eine neue Zeile verwenden. Zentrieren Sie den Text sowohl in horizontaler als auch in vertikaler Richtung. Lassen Sie zwischen den Zeilen fünf Pixel Abstand!

Hinweis zum ASCII-Datensatz: Der Datensatz ist ein zweidimensionales Array, das die gängigsten (nicht alle) ASCII-Zeichen im 6×8-Pixel Format enthält. Jede Zeile des Arrays definiert ein Zeichen durch die Angabe von 6 Bytes. Jedes dieser Bytes spezifiziert eine Spalte des 6×8 Blocks (mit MSB = 0). Die letzte Spalte ist immer leer, Sie müssen also nicht für einen horizontalen Abstand zwischen aufeinanderfolgenden Zeichen sorgen.

Aufgabe 5.4

Benutzen Sie Teile Ihrer Lösung des Aufgabenblatts 3 und entwickeln Sie einen Befehlssatz für die Nutzung über den seriellen Monitor der Arduino IDE, der folgende Funktionen zur Verfügung stellt:

- `help()`
  Listet die vorhandenen/akzeptierten Befehle mit Angabe von Information zur Nutzung dieser Befehle auf der Ausgabe des seriellen Monitors auf.
- `clearDisplay()`
  Löscht den Pufferspeicher und aktualisiert die Darstellung des LC-Displays.
- `runRotatingBarDemo()`
  Neue Funktion: Entwickeln Sie ein Verfahren das auf dem Display (zentriert) einen sich rotierenden Balken produziert: | $\rightarrow$ / $\rightarrow$ - $\rightarrow$ \ $\rightarrow \ldots$

  Verwenden Sie **3 Segmente** (ASCII-Zeichen) für die Darstellung von jedem Zustand des Balkens. Gerne können Sie auch ein vollständig grafischen Ansatz umsetzen. Verwenden Sie für die Ausführung der Funktion einen Hardware-Timer mit einer Frequenz von **10 Hz**.
- `runStudentIdDemo()`
  Erweiterte Funktion: Entwickeln Sie eine Hardware-Timer basierte Variante der in 5.2 implementierten Funktion zur abwechselnden Darstellung der Daten aller Teilnehmer Ihrer Gruppe. Wechseln Sie auch hier die Daten alle 5 Sekunden aus.
- `stopDemo()`
  Stoppt den Hardware-Timer und somit die Ausführung beider zuvor definierten Demos.

ACHTUNG: Vergessen Sie bitte nicht die Benutzereingabe auf Korrektheit zu überprüfen! Für erkannte Eingabefehler soll ein „sinnvolles Feedback“ ausgegeben werden.

Aufgabe 5.5

Eingebettete Systeme, die zur Datenaufzeichnung und/oder -wiedergabe verwendet werden, nutzen häufig externe Speichermedien. Erfordert der Anwendungsfall keine sehr hohe Schreib-/Lesegeschwindigkeit so erkauft man sich durch die Anbindung externer Speichermedien Flexibilität/Modularität (häufig sogar zu geringeren Kosten).

Ihre Aufgabe ist es, das bisher entworfene System zu erweitern. Dafür benötigen Sie den SD-Card-Slot, der sich auf der Rückseite des Displays befindet und eine vorbereitete microSD Speicherkarte. Für die Kommunikation mit der SD-Karte stellt das Display eine eigene SPI-Schnittstelle bereit. Verbinden Sie - soweit noch nicht geschehen - `MISO`, `MOSI`, `SCLK` mit dem SPI-Bus des Arduino DUE. Verbinden Sie weiterhin den `CS`-Pin der SD-Karte mit einem grundsätzlich beliebigen digitalen Ausgabe-Pin des DUE. Im Verdrahtungsvorschlag der Abb. 4

wird für `CARD_CS` bzw. `Slave-Select` der Pin **4** des DUE verwendet. **Wichtig:** Übergeben Sie die Nummer des verwendeten Anschlusspins als Argument an die Initialisierungsfunktion `SD.begin(`⟨*sce-pin*⟩`)`. Entwickeln Sie die Funktionalität für den Schreib-/Lesezugriff auf das externe Speichermedium.

Machen Sie sich zunächst mit dem Funktionsumfang der Arduino Bibliothek `SD` vertraut. Vergessen Sie nicht die Arduino SD Bibliothek korrekt in Ihr Programm einzubinden (`#include <SD.h>`). Schauen Sie sich insbesondere die folgenden Funktionen an:

```
* SD.begin(⟨sce-pin⟩)                                  → SD.begin
* SD.exists(⟨file name⟩)                               → SD.exists
* SD.open(⟨file path⟩, ⟨mode⟩)                         → SD.open
* ⟨file⟩.available()                                   → file.available
* ⟨file⟩.read()                                        → file.read
* ⟨file⟩.close()                                       → file.close
```

Auf der Ihnen ausgehändigten microSD Speicherkarte befinden sich auf der obersten Ebene des Dateisystems Textdateien mit der Endung `.txt`:

- `text1.txt`
- `text2.txt`

sowie Bilddateien mit der Endung `.img`:

- `tams.img`
- `smile1.img`
- `smile2.img`
- `smile3.img`

Erweitern Sie Ihren bisherigen Befehlssatz um folgende Funktionen für die Nutzung der SD-Karte:

- `listDirectory(`⟨*dir name*⟩`)`
  Listet den Inhalt des mit ⟨*dir name*⟩ spezifizierten Ordners als Ausgabe des seriellen Monitors auf. Die oberste Ebene des Dateisystems sollen Sie mit `/` angeben.
- `doesFileExist(`⟨*file name*⟩`)`
  Gibt Auskunft ob eine Datei ⟨*file name*⟩ auf dem Speichermedium vorhanden ist. Durch die Angabe des absoluten Pfades als Teil von ⟨*file name*⟩ können Sie auch Dateien in Unterverzeichnissen direkt erreichen.
- `outputFileToSerial(`⟨*file name*⟩`)`
  Gibt den Inhalt der mit ⟨*file name*⟩ spezifizierten Datei als Ausgabe des seriellen Monitors, in Rohform, aus.
- `outputFileToLCD(`⟨*file name*⟩`)`
  Gibt den Inhalt der mit ⟨*file name*⟩ spezifizierten Datei auf dem LC-Display, in konvertierter Form (der Dateiendung entsprechend), aus.

Beachten Sie bitte folgende Hinweise bezüglich der Dateicodierung:

- Die Textdateien enthalten **nur eine** Textzeile, abgeschlossen durch ein *newline*-Zeichen (`\n`). Beispiel: `This is some text.\n`

- Die Bilddateien enthalten zwei Zeilen ASCII-codierter Daten. Auch hier sind beide durch ein *newline*-Zeichen (`\n`) abgeschlossen.
  - Die erste Zeile enthält die Dimensionen des Bildes.
  - Die zweite Zeile enthält die eigentlichen Bildpixel: **zeilenweise angeordnet** und als eine `0` oder `1` codiert.
  - Beispiel:

```
10,10\n
0,0,0,0,0,1,1,1,1,1, . . .  ,1,1,1,1,1\n
```

Weitere Hinweise:

- Positionieren Sie die Bilder bei der Ausgabe **horizontal und vertikal zentriert** in der Anzeige des LC-Displays.
- Verwenden Sie den ASCII-Datensatz zur Darstellung des Inhalts der Textdateien. Nutzen Sie bei der Darstellung von ASCII-Zeichen die Displayfläche maximal aus. Brechen Sie Text nach der maximalen Anzahl Zeichen pro Zeile um, ohne sich um eine zusammenhängende Darstellung der Inhalte zu kümmern.
- Sollte eine Textdatei mehr Zeichen haben als auf dem LC-Display im 6×8-Format dargestellt werden können, so sollten Sie dieses entsprechend auf dem LC-Display mitteilen, beispielsweise: `TXT_SIZE_ERR`.
- *Optional* können Sie auch ein fortwährendes Scrollen implementieren, um den Text vollständig anzeigen zu können.