Aufgabenblatt 07 Termine: KW 26, KW 27

Gruppe	
Name(n)	Matrikelnummer(n)

7 Anbindung einer SD-Karte

In der Aufgabe 6 haben Sie eine Bildspeicher im SRAM des Mikroprozessors implementiert und verschiedene Funktionen programmiert, die diesen nutzen. Der Aufruf dieser Funktionen geschah hartkodiert.
Gegenstand dieser Aufgabe ist die Erweiterung der vorigen Aufgabe um eine textuelle Benutzerschnittstelle zum Aufruf dieser Funktionen. Zusätzlich werden Sie den Zugriff auf ein externes Speichermedium, eine SD-Karte, implementieren und auch diese Funktionalität in das Benutzerinterface integrieren.

Aufgabe 7.1 Entwicklung eines String-Parsers

Entwickeln Sie einen String-Parser, welcher ein über das Eingabefeld des seriellen Monitors abgeschicktes Kommando entgegennimmt und dieses auf Korrektheit überprüft. Bei einem korrekt erkannten Kommando veranlassen Sie die Ausführung der Funktion, andernfalls geben Sie, soweit möglich, eine möglichst aussagekräftige Fehlermeldung auf der seriellen Konsole aus.
Der Befehlssatz soll es erlauben, folgende z. T. bereits in Aufgabe 6 entwickelte Funktionen aufzurufen:

- `help()`
  Listet die gültigen Befehle mit Angabe von Information zur Nutzung dieser Befehle in der Ausgabe des seriellen Monitors auf.
- `clearDisplay()` `clear()`
  Löscht den Pufferspeicher und aktualisiert die Darstellung des Displays.
- `runRotatingBarDemo()` `runRBD()`
  Starten Sie Ihre in Aufgabe 6.4 entwickelte `RotatingBarDemo()`.
- `runStudentIdDemo()` `runID()`
  Starten Sie Ihre in Aufgabe 6.6 entwickelte `StudentIdDemo()`.

- `stopDemo()` `stop()`
  Stoppen Sie die Ausführung der ggf. gerade laufenden Demo (RotatingBarDemo, StudentIdDemo), z. B. durch Stoppen des jeweiligen Hardware-Timers.

ACHTUNG: Vergessen Sie nicht, die Benutzereingabe auf Korrektheit zu überprüfen!
Für erkannte Eingabefehler soll ein „sinnvolles Feedback“ ausgegeben werden. Berücksichtigen Sie bei der Entwicklung des Parsers eine leichte Erweiterbarkeit des Befehlssatzes.

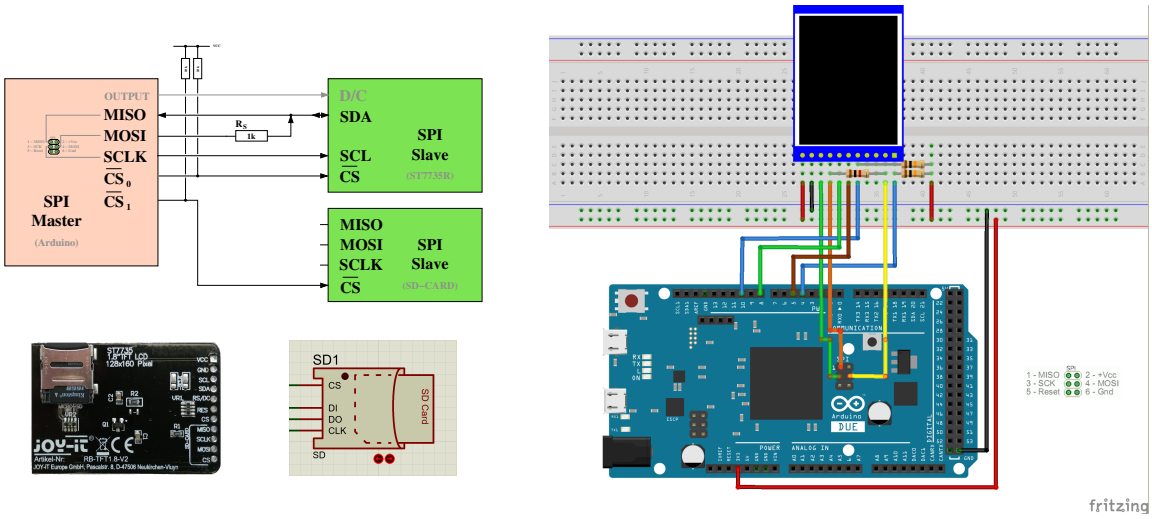


Abbildung 1: Vorschlag für die Verdrahtung des Versuchsaufbaus. Bitte **ergänzen** Sie die Verdrahtung des **SD-Slots** der Abbildung oben links **und** vergessen Sie beim Hardwareaufbau ebenfalls nicht, den auf dem Displayboard integrierten SD-Slot anzuschließen. Das ist in Abb. 1 noch **nicht** vollständig geschehen!

Aufgabe 7.2 Einbindung einer SD-Karte

Eingebettete Systeme, die zur Datenaufzeichnung oder auch zur Anzeige von Daten verwendet werden, nutzen häufig externe Speichermedien. Erfordert der Anwendungsfall keine sehr hohe Schreib-/Lesegeschwindigkeit, so ermöglicht die Anbindung externer Speichermedien zusätzliche Flexibilität.

Ihre Aufgabe ist es, das bisher entworfene System um ein solches externes Speichermedium zu erweitern. Dafür benötigen Sie in unserem Falle den Zugriff auf eine SD-Karte.

Sie finden den SD-Card-Slot der Rückseite des Displays, in den bereits eine vorbereitete microSD Speicherkarte eingesetzt ist. Für die Kommunikation mit der SD-Karte stellt das Breakoutboard des Displays eine eigene SPI-Schnittstelle bereit.
Mit der SD-Karte kommunizieren Sie ebenfalls über die SPI-Schnittstelle.
Vervollständigen Sie - soweit noch nicht geschehen - den Verdrahtungsvorschlag der Abb. 1 um die Verbindungen des SD-Interfaces mit dem SPI-Bus des Arduinos. Verbinden Sie weiterhin den `CS`-Pin der SD-Karte mit einem grundsätzlich beliebigen digitalen Ausgabe-Pin des Arduino.

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
```

Bitte bedenken Sie, dass die Arduino SD Bibliothek für Lese- und Schreiboperationen einen 512 byte-Puffer anlegt. Sollte Ihr Sketch das verfügbare SRAM (Speicherort für Variablen, dynamische Datenstrukturen (Heap) und des Stacks) bereits nahezu auslasten, kann es durch den Zusatzbedarf zu unvorhersehbarem Programmverhalten führen (siehe hierzu auch: www.arduino.cc/en/Tutorial/Foundations/Memory). Diese Situation kann insbesondere auf dem Arduino MEGA2560 aufgrund seines – verglichen mit dem Arduino DUE – kleinen SRAMs schnell eintreten. Ohne das Vorhandensein eines Betriebssystems werden Situationen wie „Stack overruns Heap“oder „Heap overruns Stack“ nicht erkannt.

Auf der für die Übungen vorbereiteten Speicherkarte bzw. des vorbereiteten Images der 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⟩)` `ls(⟨dir name⟩)`
  Listet den Inhalt des mit ⟨*dir name*⟩ spezifizierten Ordners in der Ausgabe des seriellen Monitors auf. Die oberste Ebene des Dateisystems sollen Sie mit `/` angeben.
- `doesFileExist(⟨file name⟩)` `exist(⟨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⟩)` `catSer(⟨file name⟩)`
  Gibt den Inhalt der mit ⟨*file name*⟩ spezifizierten Datei in der Ausgabe des seriellen Monitors in Rohform aus.
- `outputFileToLCD(⟨file name⟩)` `catTFT(⟨file name⟩)`
  Gibt den Inhalt der mit ⟨*file name*⟩ spezifizierten Datei auf dem LC-Display, in konvertierter Form (der Dateiendung entsprechend), aus.

  Hinweis: Der ST7735R-Controller unseres Displays ignoriert nach nicht an ihn gerichteten Datenverkehr auf dem SPI-Bus das erste wieder an ihn gerichtete Kommando. Verwenden Sie also bitte, wie auch schon im Beispielsketch (TFT_template_mCS.ino) aus Aufgabe 6 z.B. in der Funktion `TFTwriteWindow(..)` gezeigt, einen obsoleten Befehl – dort der NOP-Befehl –, um den Controller vorzubereiten.

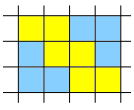
Beachten Sie bitte folgende Hinweise bezüglich des Aufbaues der auf der microSD abgelegten Dateien:

`.txt` Die Textdateien enthalten **nur eine** Textzeile, abgeschlossen durch ein *newline*-Zeichen (`\n`). Beispiel: `This is some text.\n` (siehe auch: `text1.txt`).

`.img` Die Bilddateien enthalten zwei Zeilen ASCII-codierter Daten. Auch hier sind beide durch ein *newline*-Zeichen (`\n`) abgeschlossen.

- Die erste Zeile enthält die Dimension (Breite, Höhe) des Bildes.
- Die zweite Zeile enthält die eigentlichen Bildpixel: **bildzeilenweise fortlaufend angeordnet** und als `0` (Hintergrundfarbe) oder `1` (Vordergrundfarbe) codiert.
- Beispiel: (gelb - Vordergrundfarbe, blau - Hintergrundfarbe)

```
4,3\n
1,1,0,0,0,1,1,0,0,0,1,1\n
```



(siehe auch: `tams.img`)

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

den 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.