Aufgabenblatt 07 Termine: KW 24, KW 25

Gruppe	
Name(n)	Matrikelnummer(n)

7 Implementierung einer textuellen Benutzerschnittstelle und Anbindung einer SD-Karte

Die serielle UART-Schnittstelle 0 des Arduino wurde unter Verwendung der Arduino `Serial`-Library erstmals auf Aufgabenblatt 2 zur Ausgabe von Programminformationen auf die serielle Konsole der Arduino-IDE eingesetzt. Die UART-Schnittstelle erlaubt allerdings nicht nur eine unidirektionale Datenübertragung, sondern sie erlaubt auch eine bidirektionale Kommunikation. So verfügt die Arduino `Serial`-Library auch über Funktionen, die das Lesen vom seriellen Port erlauben.

Zusätzlich zu den bereits bekannten Funktionen sind hier insbesondere die `Serial.available`- und die `Serial.read`-Funktion interessant:

```
* Serial.begin(⟨speed⟩)                    → Serial.begin
* Serial.available()                       → Serial.available
* Serial.read()                            → Serial.read
* Serial.write(⟨arg⟩)                      → Serial.write
* Serial.print(⟨arg⟩)                      → Serial.print
* Serial.println(⟨arg⟩)                    → Serial.println
```

Machen Sie sich mit den neuen Funktionen anhand des folgenden Beispielcodes vertraut:

```
#define INPUT_LENGTH 127

char input_buffer[INPUT_LENGTH] = { 0 };
uint8_t buffer_pos = 0;

void setup() {
  Serial.begin(9600);
  Serial.print("your input: ");
}
```

```
void loop() {
  while (Serial.available()) {
      char ch = Serial.read();    //read char
      Serial.write(ch);           //echo to serial monitor

      if ((ch != 0x0A) && (ch != 0x0B) && (ch != 0x0C) && (ch != 0x0D)) {
          input_buffer[buffer_pos++] = ch;
          if (buffer_pos == INPUT_LENGTH) {
              buffer_pos = 0;
              Serial.println("WARNING: Input buffer overflow!");
          }
      }
      else {    // Output input string
          input_buffer[buffer_pos] = '\0';
          Serial.print("read in line: ");
          Serial.println(input_buffer);
          buffer_pos = 0;
          input_buffer[buffer_pos] = '\0';
      }
  }
}
```

Bitte beachten Sie, dass sich die seriellen Monitore der Arduino-IDEs untereinander und das virtuelle Proteus-Terminal sich wiederum leicht unterschiedlich zu einander verhalten:

- Die seriellen Monitore der Arduino-IDEs der Versionen < 2.0 übertragen die Zeichen erst nach Eingabe von **ret**, bei der Version 2.0 erst nach Eingabe von **strg + ret**. Soll die Eingabe zur Erkennung des Zeilenendes zusätzlich durch ein **LF** (line feed – 0x0A) oder **CR** (carriage return – 0x0D) abgeschlossen werden, wählen Sie bitte im `Serial Monitor` die Option `New Line` oder `Carriage Return`.
- Das virtuelle Terminal von Proteus 8 überträgt jedes eingetippte Zeichen sofort und die Eingabe von **ret** resultiert in der Übertragung von **LF** (0x0A), während ein **strg + ret** die Übertragung von **CR** (0x0D) bewirkt. Wenn Sie die eingegebenen Zeichen vom Terminal selbst angezeigt bekommen möchten, aktivieren Sie bitte den Echo-Mode.

Nutzen Sie auch die Dokumentation der `Serial`-Bibliothek und auch die Dokumentation ihrer Funktionen (s. o.).

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()`
  Löscht den Pufferspeicher und aktualisiert die Darstellung des Displays.
- `runRotatingBarDemo()`
  Starten Sie Ihre in Aufgabe 6.4 entwickelte `RotatingBarDemo()`.
- `runStudentIdDemo()`
  Starten Sie Ihre in Aufgabe 6.6 entwickelte `StudentIdDemo()`.
- `stopDemo()`
  Stoppen Sie die Ausführung der ggf. gerade laufenden Demo (RotatingBarDemo, StudentIdDemo), z. B. durch Stoppen des jeweiligen Hardware-Timers.

Die Namen der Befehle dürfen Sie, insbesondere für Proteus, gerne (tipp)ökonomischer wählen.

ACHTUNG: Vergessen Sie bitte 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.

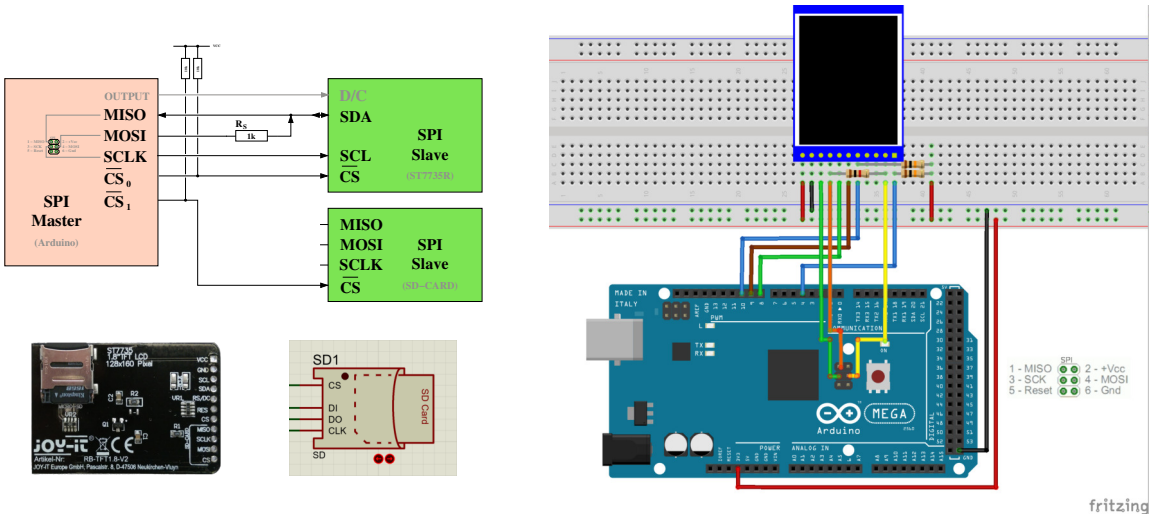


Abbildung 1: Vorschlag für die Verdrahtung des Versuchsaufbaus. SD-Slot noch nicht vollständig angeschlossen.

Aufgabe 7.2 Einbindung einer SD-Karte

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

Ihre Aufgabe ist es, das bisher entworfene System zu erweitern. Dafür benötigen Sie ein SD-Slot-Device, dass Sie bei Proteus unter `'Pick Devices'` mit dem Schlüsselwort `"Virtual SD"` finden und ein Image einer microSD Speicherkarte. Geben Sie bitte unter `'Edit Component'`

(Rechtsklick auf SD-Slot - `'Edit Properties'`) bei `'Card Image File'` den Pfad zum extrahierten Image (SD4G_1GB.mmc) an. Mit der SD-Karte kommunizieren Sie ebenfalls über die SPI-Schnittstelle, wobei die Label der Anschlusspins des SD-Devices mit `DI` für `Data IN`, `DO` für `Data Out` und `CLK` für `Clock` stehen.
Vervollständigen Sie - soweit noch nicht geschehen - den Verdrahtungsvorschlag der Abb. 1 um die Verbindungen des SD-Interfaces mit dem SPI-Bus des Arduino MEGA. Verbinden Sie weiterhin den `CS`-Pin der SD-Karte mit einem grundsätzlich beliebigen digitalen Ausgabe-Pin des Arduino. Im Verdrahtungsvorschlag der Abb. 1 wird für `CARD_CS` bzw. `Slave-Select` der Pin **4** des Arduinos verwendet, um die Kompatibilität zu aufsteckbaren Erweiterungsshiels mit SD-Slot zu gewährleisten, da diese i.d.R. Pin 4 verwenden.
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`). Ohne das Vorhandensein eines Betriebssystems werden Situationen wie „Stack overruns Heap“oder „Heap overruns Stack“ nicht erkannt und führen zum o. g. Verhalten.

Auf dem für die Übungen vorbereiteten Image 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*⟩`)`
  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*⟩`)`
  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 in der 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.

  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.

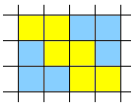
Beachten Sie bitte folgende Hinweise bezüglich des Aufbaues der auf der microSD abgelegten Dateien:

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

`.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
```



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.