64-211 Übung Eingebettete Systeme

Aufgabenblatt 5 Termine: 08.05./11.05.

Gruppe	
Name(n)	Matrikelnummer(n)

Flüssigkristalldisplays 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 Nokia 5110-kompatibles grafisches LC-Display einsetzen, um damit grundlegende Funktionalität zur visuellen Ausgabe zu implementieren.

Die Kommunikation mit dem Display findet über eine serielle Schnittstelle (**SPI**) statt:

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

Die Schnittstelle wird vom Displaycontroller (Philips PCD8544) bereitgestellt. Setzen Sie sich mit dem Datenblatt des Displaycontrollers auseinander. Beachten Sie insbesondere Informationen zu den Anschlusspins sowie den Funktionen des Befehlssatzes. Setzen Sie sich ferner mit der Adressierung und der Initialisierung auseinander.

ACHTUNG: In der Übung kommen Displays mit **zwei unterschiedlichen Versionen von Platinen** zum Einsatz! In Abbildung 1 sind die Anschlüsse beider Versionen (**Rot** und **Blau**) der Platinen dargestellt. Beachten Sie, dass die beiden Versionen **hinsichtlich der Verdrahtung inkompatibel** zueinander sind. Führen Sie die notwendige Verdrahtung des LC-Displays der jeweiligen Version der Platine entsprechend auf dem Steckbrett durch.

Für die Lösung der Aufgabe wird empfohlen den Versuchsaufbau gemäß Abbildung 2 zu verdrahten. Beachten Sie, dass es sich bei dem Verdrahtungsvorschlag um die **Rote** Version der Platine handelt! Ferner ist im Verdrahtungsvorschlag bereits das Arduino Wireless SD Shield enthalten. Diese Zusatzplatine wird erstmalig für Aufgabe 5.4 benötigt, kann jedoch von Anfang an auf das Arduino Due Board aufgesteckt werden. Beachten Sie bei der Verdrahtung folgende Hinweise:

- Verbinden Sie `VCC` mit **3,3 V** und stellen Sie zusätzlich die Masseverbindung her.
- Verbinden Sie `SCE` bzw. `CS` mit einem der freien Hardware-gesteuerten `Slave-Select` Anschlusspins des Arduino Due: **10** oder **52**.
- Verwenden Sie für die Verbindung der Anschlüsse `RST` und `D/C` beliebige digitale Anschlusspins des Arduino Due.

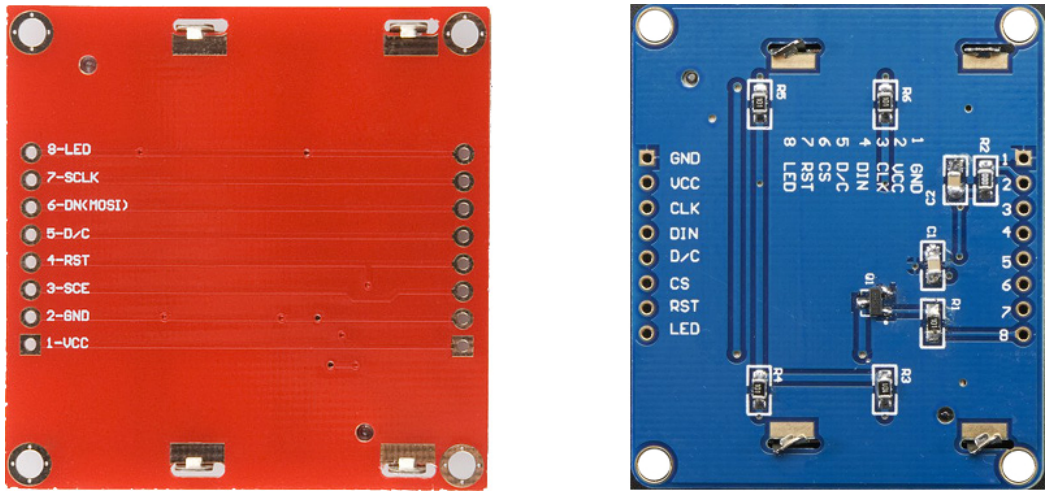

Abbildung 1: Anschlusspins beider Versionen der LC-Display Platinen.

- Verbinden Sie `SCLK` bzw. `CLK` und `DN(MOSI)` bzw. `DIN` mit den entsprechenden Anschlusspins der **ersten SPI Schnittstelle** (siehe Arduino Due Pinbelegung).
- Die Hintergrundbeleuchtung (`LED`) muss nicht angeschlossen werden. Sollten Sie es dennoch tun wollen, stellen Sie eine Verbindung mit einem beliebigen (evtl. PWM-fähigen) digitalen Anschlusspin des Arduino Due her. Alternativ können Sie `LED` auch direkt an **3,3 V** anschließen

Aufgabe 5.1

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>`). Bei unserem Arduino Due besitzen die Funktionen `SPI.beginTransaction` und `SPI.transfer` noch einen weiteren Parameter[1], den *Slave-Select* bzw. *Chip-Select* Anschlusspin.

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

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

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

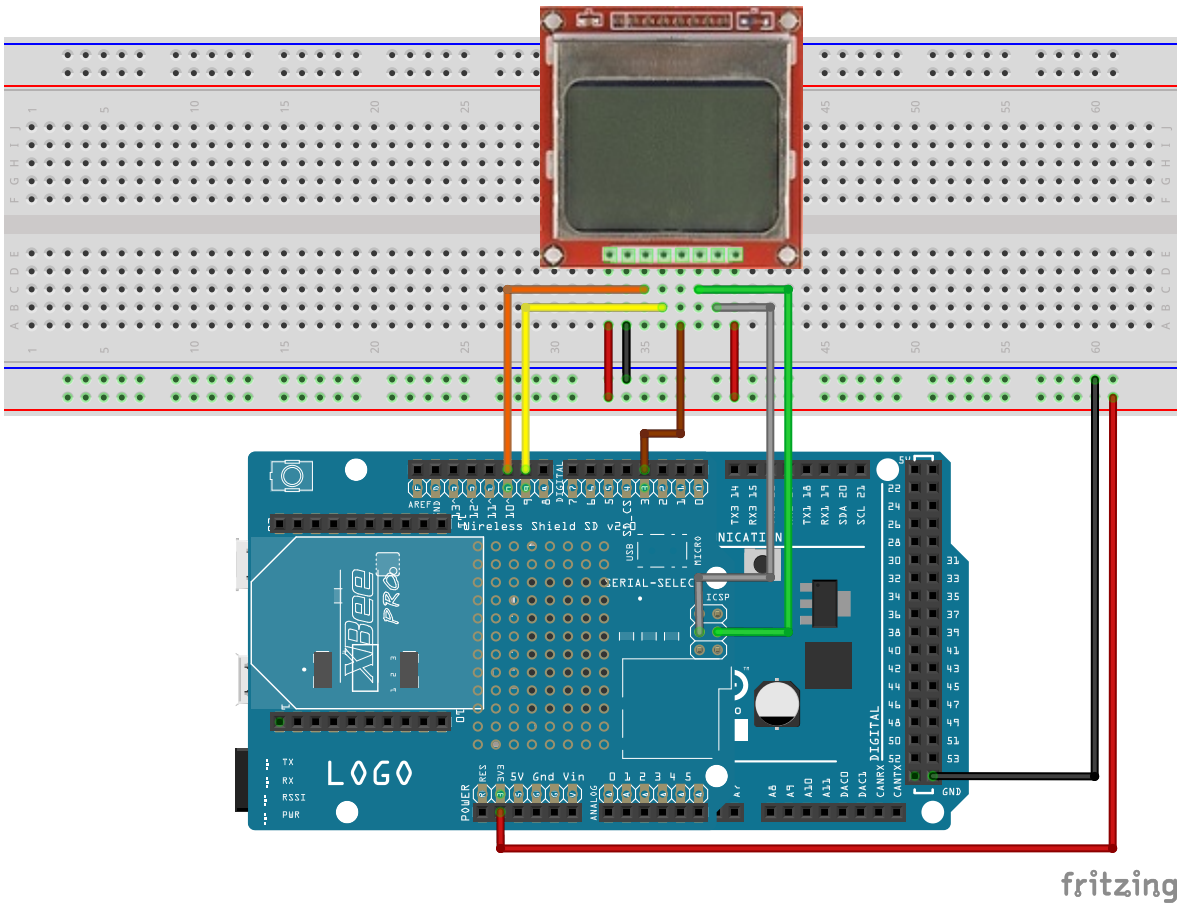

Abbildung 2: Vorschlag für die Verdrahtung des Versuchsaufbaus.

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 LC-Display wird eine 500ms lange `RESET`-Phase empfohlen.
- 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 (siehe dazu das Datenblatt, S. 15-16):

  1. `0x21` : FUNCTION SET (extended instruction set)
  2. `0x13` : SET BIAS (1:48 MUX, 1/8 BIAS)
  3. `0xB4` : SET CONTRAST (V_OP = 6,18 V)
  4. `0x04` : SET TEMPERATURE COEFFICIENT
  5. `0x20` : FUNCTION SET (normal instruction set)
  6. `0x0C` : SET DISPLAY MODE (normal)
  7. `0x40` : SET Y COORDINATE = 0
  8. `0x80` : SET X COORDINATE = 0

Hinweise zu Punkt 2:

- Das LC-Display besitzt eine Auflösung von 48 × 84 Pixeln (Zeilen × Spalten). Eine Adressierung einzelner Pixel ist mit dem Befehlssatz des Displaycontrollers jedoch nicht möglich (siehe Datenblatt, S. 9-10). Die 48 Pixelzeilen sind in 6 Gruppen (≙ Textzeilen; *Bank*) unterteilt. Während sich die Adressen entlang der x-Achse im Bereich 0…83 bewegen ist der Adressbereich entlang der y-Achse auf 0…5 reduziert.
- 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 ⟨*x*⟩,⟨*y*⟩-Position das Pixel ⟨*value*⟩∈(`0`, 1) setzt: `setPixel(int x, int y, int 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 5.2

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)`

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)`

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 5.3

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.
- `setContrast(`⟨*value*⟩`)`
  Neue Funktion: Entwickeln Sie eine Funktion zur Einstellung des Kontrastes des LC-Displays. Verwenden sie für ⟨*value*⟩ einen Wertebereich von $0.0 \ldots 1.0$.
- `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: | → / → - → \ → ...

  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 5.4

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 das "'Arduino Wireless SD Shield"' zuzüglich einer vorbereiteten microSD Speicherkarte. Entwickeln Sie die Funktionalität für den Schreib-/Lesezugriff auf das externe Speichermedium.

Sofern nicht bereits geschehen, stecken Sie das Wireless SD Shield gemäß dem in Abbildung 2 dargestellten Versuchsaufbau auf den Arduino Due auf. Platzieren Sie die microSD Speicherkarte im entsprechenden Slot. Beachten Sie bitte folgenden Hinweis zur Verdrahtung:

- Das Arduino Wireless SD Shield sieht **Anschlusspin 4** exklusiv als `Slave-Select` bzw. `Chip-Select` Anschlusspin vor. **Wichtig:** Übergeben Sie die Nummer dieses Anschlusspins als Argument an die Initialisierungsfunktion `SD.begin(`⟨*sce-pin*⟩`)`.

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 14 Zeichen 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.