64-211 Übung Eingebettete Systeme

Aufgabenblatt 05 Termine: KW 22

Gruppe	
Name(n)	Matrikelnummer(n)

5 Arbeiten mit realer Hardware

Nachdem bislang alle Aufgaben der Übungen mit dem Proteus Simulator erprobt wurden, wird jetzt, nachdem die Übungen nun in Präsenz durchgeführt werden, mit realer Hardware gearbeitet.
Die bisherige Codeentwicklung war auf den Arduino MEGA2560, der auf dem ATmega2560-Prozessor, einem 8-bit AVR-Prozessor, basiert, ausgerichtet. Für die Präsenzübungen werden wir aber das Arduino DUE-Board verwenden, das einen Atmel SAM3X8E-Prozessor, einen 32-bit ARM Cortex-M3 RISC-Prozessor, einsetzt. Die Unterschiede der Architekturen verbirgt die Arduino-IDE weitestgehend vor dem Programmierer. Einzig wenn Besonderheiten der Architekturen ausgenutzt werden sollen, hat der Programmierer aktiv zu werden. Für die Aufgaben zu ES betrifft es – wenn nicht unbedingt die „Enhanced SPI-Library“ des Due eingesetzt werden soll – lediglich die Verwendung der Timer. Aufgrund der höheren Anzahl der Timer ist der Due-Programmierer hier flexibler und vor allem existiert eine eigene Timer-Library für den Due, die zu verwenden ist. Hier ist der bisherige Code entsprechend zu erweitern, so dass er möglichst für beide Zielarchitekturen kompilierbar ist.

Die `DueTimer` Bibliothek können Sie von der Webseite der Veranstaltung herunterladen: DueTimer Bibliothek. Wie bereits die Timer-Bibliotheken für den ATmega2560 müssen Sie die `DueTimer` Bibliothek in den dafür von der Entwicklungsumgebung vorgegebenen Ordner verschieben. Dieser Ordner befindet sich im Home-Verzeichnis des Benutzers.
Der Pfad des Ordners ist unter Linux: `~/Arduino/libraries`
und unter Win10/11: `~\Documents\Arduino\libraries`

Aus Ihren bisherigen Proteus-Firmware Projekten müssen Sie in einen Arduino Sketch ableiten. Haben Sie Ihre Proteus-Projekte unter Verwendung des Proteus *New Project Wizards* aufgesetzt, haben Sie ihren Code üblicherweise unter „`main.ino`“entwickelt. Kopieren Sie ihren Proteus-Code der „`main.ino`“ in eine Textdatei „<name>“ mit der Extension „`.ino`“. Diese Datei muss in einem gleichnamigen Verzeichnis, hier also „`name`“, liegen. Entfernen Sie aus Ihrem neuen Arduino-Sketch „<name>`.ino`“ ggf. die durch den Proteus *New Project Wizard* hinzugefügten Proteus-Spezifika. Dieses ist die Präambel, beginnend bei „`/* Main.ino file generated by`

`New Project wizard`“ und endend bei „`//---CONFIG_END---`“.
Entsprechend ist noch aus der setup-Routine der Funktionsaufruf „`peripheral_setup()`“ und aus der Main-loop der Aufruf der „`peripheral_loop()`“ zu entfernen. Jetzt können Sie Ihren Arduino-Sketch mit der Arduino-IDE übersetzen.

Es wird allerdings dringend nahegelegt, den Code weiterhin für beide Zielarchitekturen kompilierbar zu halten, was mittels Präprozessordirektiven möglich ist. Grundsätzlich ist dafür folgendes zu beachten:

- Verwenden Sie keine architekturspezifischen Zusatzfunktionen, wie beispielsweise die Enhanced SPI-Library des Due.
- Binden Sie beispielsweise die Timer-Library mittels Präprozessordirektiven ein.
  Benennen Sie das Due-Timerobjekt entsprechend der Mega-Notation. Auf diese Weise sind die Methodenaufrufe im Code identisch.
  Der Beispielsketch DueMegaTimerTemplate.ino zeigt es exemplarisch.

Nur so kann der Code weiterhin unter Proteus für den ATmega2560 entwickelt und validiert werden und anschließen auf dem Arduino DUE ausgeführt werden; über die für das Kompilieren gewählte Zielarchitektur wird die Codeauswahl gesteuert.

Auf den TAMS-Laborrechnern im Raum F304 bzw. F325 starten Sie die Arduino-IDE mit dem Kommando: $tamsSW/arduino-1.8.19/arduino

Aufgabe 5.1 Erproben der Aufgabe 4.3 des vorigen Aufgabenblattes auf realer Hardware

Bauen Sie den Versuchsaufbau entsprechend des Proteus Schematics zu Aufgabe 4.3 mit den Komponenten des ES-Arduino Kits auf. Rufen Sie sich dafür insbesondere die Hinweise aus Aufgabe 1 zu den dortigen Abbildungen 1 – 4 in Erinnerung. Entwickeln sie, wie oben beschrieben, den Arduino-Sketch aus dem Proteus-Projekt, und nutzen Sie die Arduino-IDE zum Kompilieren des Sketches und zum Hochladen der Firmware auf den Mikrocontroller.

Der praktische Aufbau könnte ähnlich wie in Abb. 1 dargestellt aussehen.

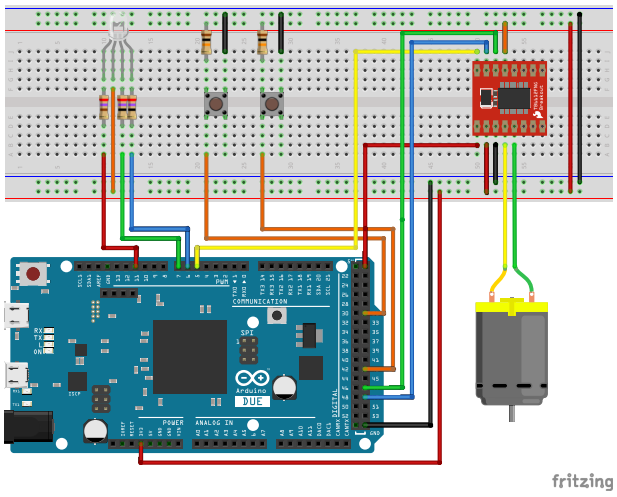

Abbildung 1: Vorschlag für eine mögliche Verdrahtung des Versuchsaufbaus.

5.1 Elektrische Ansteuerung eines Displays

Flüssigkristalldisplays (LCDs) sind ein häufig verwendetes Ausgabegerät für eingebettete Systeme. Entsprechend des Bedarfes werden einfache Zeichen-Displays oder auch grafische Displays eingesetzt. Für die späteren Aufgaben der Übungen zu ES werden Sie ein grafisches 1.8" TFT-LCD (Thin-film-transistor liquid-crystal display) verwenden, um darauf grundlegende Funktionalitäten zur visuellen Ausgabe zu implementieren.

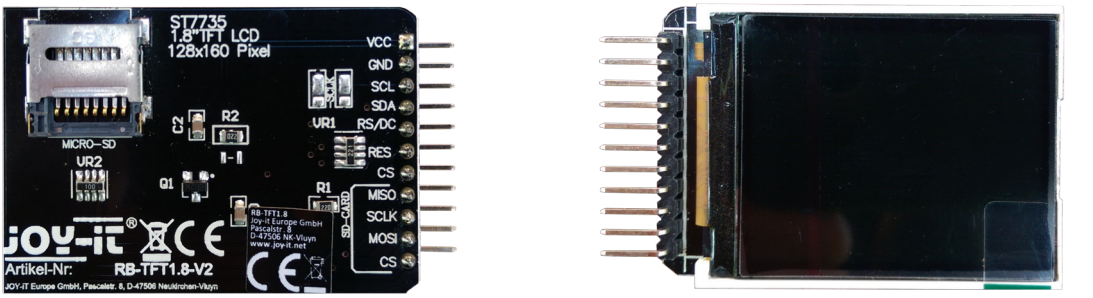

Abbildung 2: 1.8" TFT-Display mit zusätzlichem SD-Card-Slot.

Die Schnittstelle dieses Displays wird durch den 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 des Displaycontrollers auf dem Breakout-Board des in den ES Übungen verwendeten Displays (siehe Abb. 2) ist die serielle 4-Draht Schnittstelle hardwareseitig fest konfiguriert. Abb. 3 zeigt das vereinfachte Timing-Diagramm dieser Schnitstelle.

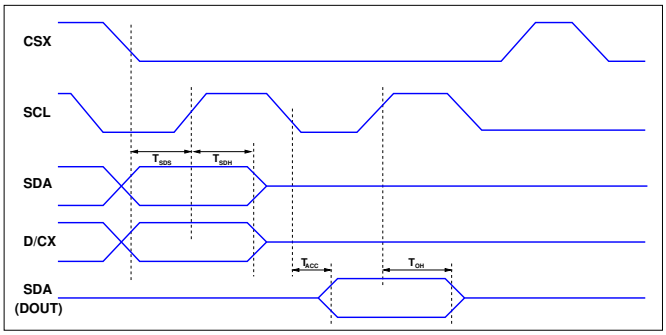

Abbildung 3: Vereinfachtes Timing-Diagramm der 4-Draht Schnittstelle des ST7735R;
Leitungen: `CSX` – Chip-Select, `SCL` – Serial Clock Line (Taktsignal),
`SDA` – Serial Data Line (Datenleitung), `D/CX` – Data/Command;
Timingangaben: T_{SDS} – Data setup time, T_{SDH} – Data hold time,
T_{ACC} – Access time, T_{OH} – Output disable time
(vgl. Datenblatt ST7735R, S.25)

Aufgabe 5.2 Diskussion eines Anschlussbeispiels

Diskutieren Sie das in Abb. 4 skizzierte Beispiel zum Anschluss des Displays an einen Arduino. Für erste Überlegungen gehen Sie davon aus, dass für die Anschlüsse des Displays entsprechend Abb. 3 seitens des Arduinos normale digitale I/O-Pins verwendet werden.

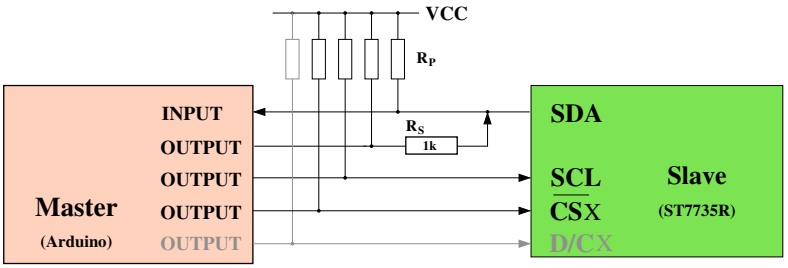

Abbildung 4: Anschlussvorschlag des Displays mit ST7735R-Controller

- Wäre diese Beschaltung grundsätzlich möglich und sinnvoll?
- Welche Bedeutung hätte dann der Widerstand R_S?
- Könnte er auch fortgelassen werden?
- Was können Sie über die Dimensionierung von R_S aussagen?
- Welche Bedeutung haben die Pullup-Widerstande R_P?
- Können/sollten Pullup-Widerstande fortgelassen werden?
  Und falls ja, welche und unter welchen Voraussetzungen?

Die in engebetteten Systemen eingesetzten Prozessoren stellen i. d. R. verschiedene serielle Schnittstellen, die für die Ansteuerung von Sensoren und Aktuatoren üblicherweise benötigt werden, bereits zur Verfügung. So stellen auch die Prozessoren der Arduino-Boards folgende **serielle Protokolle** bereit:

- **UART**: **U**niversal **A**synchronous **R**eceiver/**T**ransmitter
  Beispielsweise **RS-232**: Ein Kommunikationspartner pro Schnittstelle, asynchroner Datentransfer
- **I²C**: **I**nter-**I**ntegrated **C**ircuit
  Serieller Bus, (multi-)**Master/Slave** Kommunikation, synchroner Datentransfer
- **SPI**: **S**erial **P**eripheral **I**nterface
  Serieller Bus, **Master/Slave** Kommunikation, synchroner Datentransfer

Aufgabe 5.3 Serielle Busprotokolle

Machen Sie sich mit den o. g. seriellen Schnittstellen und ihren (Bus)Protokollen vertraut. Die referenzierten Wikepedia-Seiten bieten schon einen guten Überblick.

Lässt sich eine dieser Schnittstellen zur Ansteuerung des Displays über die serielle 4-Draht Schnittstelle des ST 7735R-Displaycontrollers (siehe Abb. 3) verwenden? Sie werden feststellen, dass keine dieser Schnittstellen auf den ersten Blick mit der seriellen 4-Draht Schnittstelle des ST 7735R-Displaycontrollers kompatibel ist. Berücksichtigen Sie bei Ihrer Einschätzung nicht nur die elektrischen Eigenschaften Hardwareverbindungen, sondern auch die Protokolle der Schnittstellen.
Welche dieser Schnittstellen halten sie ggf. für geeignet, das in den ES-Übungen verwendete Display anzusteuern. Begründen Sie Ihre Entscheidung.