Handreichung PhC - Arduino Datenbus

1. Kurzvorstellung

Ein System aus zwei Arduinos (Sender und Empfänger), die den Datenaustausch über einen Datenbus simulieren sollen.

2. Einordnung in die Lehrpläne

Da die technische Informatik in der Oberschule und am beruflichen Gymnasium wenig bis gar nicht thematisiert wird, werden hierfür auch keine Lernbereiche angegeben. In niedrigeren Klassen könnte das fertige System höchstens zur Simulation der Datenströme verwendet werden , ohne weiter auf die Umsetzung einzugehen.

Gymnasium Klassenstufen 11/ 12

Lernbereich 8B:

- „Einblick gewinnen in die Prozessdatenverarbeitung – Signal, Daten, Datentransport“
- „Anwenden der Kenntnisse über die Ansteuerung paralleler und serieller Schnittstellen“
- „Vor- und Nachteile paralleler und serieller Datenübertragung“

3. Lernziele

Kognitiv

Die Schülerinnen und Schüler...

- lernen eine Möglichkeit auf Bit-Ebene Daten zwischen Informatiksystemen auszutauschen kennen. (parallel)
- lernen die Bedeutung von Taktfrequenz und Busbreite für die Geschwindigkeit des Datenaustausches kennen.
- lernen eine Möglichkeit Zeichen zu kodieren kennen (ASCII) und erkennen den Sinn der Kodierung für die Datenübertragung.
- vertiefen ihre Programmierkenntnisse in einer imperativen Programmiersprache.

Affektiv

- erkennen den Zusammenhang zwischen gelernten Inhalten (Binärzahlen, Zeichenkodierungen) und der Umsetzung in der Hardware.
- erkennen die Möglichkeiten arbeitsteiligen Vorgehens in der Softwareentwicklung.

Psychomotorisch

- programmieren (mithilfe vorgegebener Methoden) Software, die Textaustausch zwischen Arduinos ermöglicht.
- verwenden einfache elektronische Bauteile zur Realisierung einer Schaltung.

4. Voraussetzungen

Fachlich

- grundlegende Kenntnisse in der Algorithmierung und Programmierung
  - Sequenz, Selektion, Zyklus (Klasse 9)
  - Kenntnisse in einer höheren Programmiersprache (Datentypen, grundlegende Syntax)
- Kenntnis über für das Projekt nötige Funktionen der Arduino Sprache (vorhergehend oder im Projekt integrierte Aneignung)
  - grundlegende Programmstruktur `setup()` und `loop()`
  - `delay()` `pinMode()` `digitalWrite()` `digitalRead()` `bitWrite()` `bitRead()`
- grundlegende Kenntnisse über Gleichstromschaltkreise

Materiell

- Arduino/ Funduino Mega (oder andere Modelle, die I2C unterstützen)
- beliebiger anderer Arduino programmierbarer Microcontroller mit mindestens 8 GPIO Pins (hier D1 mini (ESP 8266))
- 17x male-male, 4x male-female Jumperkabel
- 8x LED
- Steckbrett
- Vorwiderstand für LEDs (hier 330 Ω)
- I²C Display
- Stromversorgungen

Um das Kabelchaos etwas zu reduzieren könnten auch Steckbretter für Sender und Empfänger vorbereitet werden, die dann am Ende mithilfe eines Ethernet Kabels und RJ45 Buchsen mit sogenannten Breakout Boards auf den Steckbrettern verbunden werden. (Internetsuche nach RJ45 Breakout Board, bspw. SparkFun)

Technisch

- Computer mit Arduino IDE
  - installierte Bibliotheken für Display und Microcontroller
  - Beispielhaft für verwendeten D1 mini und Display:
    - in IDE: Datei → Voreinstellungen → Zusätzliche Beadverwalter-URLs

        - http://arduino.esp8266.com/stable/package_esp8266com_index.json
      - bestätigen → Werkzeuge → Board: ... → Boardverwalter
      - Suche nach esp8266 → esp8266 by ESP8266 Community → installieren, Boardverwalter schließen
      - Werkzeuge → Board: ... → ESP8266 Boards → LOLIN(WEMOS) D1 R2 & mini
      - Werkzeuge → Bibliotheken verwalten → suche LiquidCrystal I2C → LiquidCrystal I2C by Marco Schwartz → installieren
  - Falls andere Komponenten benutzt werden sollen, gibt es für alle Möglichen Dinge Anleitungen über die Suchmaschine der Wahl zu finden.

5. Kurzdarstellung

Hinweise zum Nachbau/ zur Verwendung

- die zu übertragende Nachricht wird im Code des Senders im Array `message[]` festgelegt
- falls andere Pins verwendet werden sollen, weil beispielsweise abweichende Mikrocontroller verwendet werden, müssen diese im jeweiligen Code als Array `dataPins[]` eingegeben werden
- falls die Übertragungsrate verändert werden soll, (Standard eine Nachricht pro Sekunde) muss dies in beiden Codes in der `delayTime` getan werden
- eventuell ist es nötig am LCD Display Einstellungen vorzunehmen (beispielsweise Einstellung des Kontrastes über eine Schraube auf der Rückseite)

Projektidee

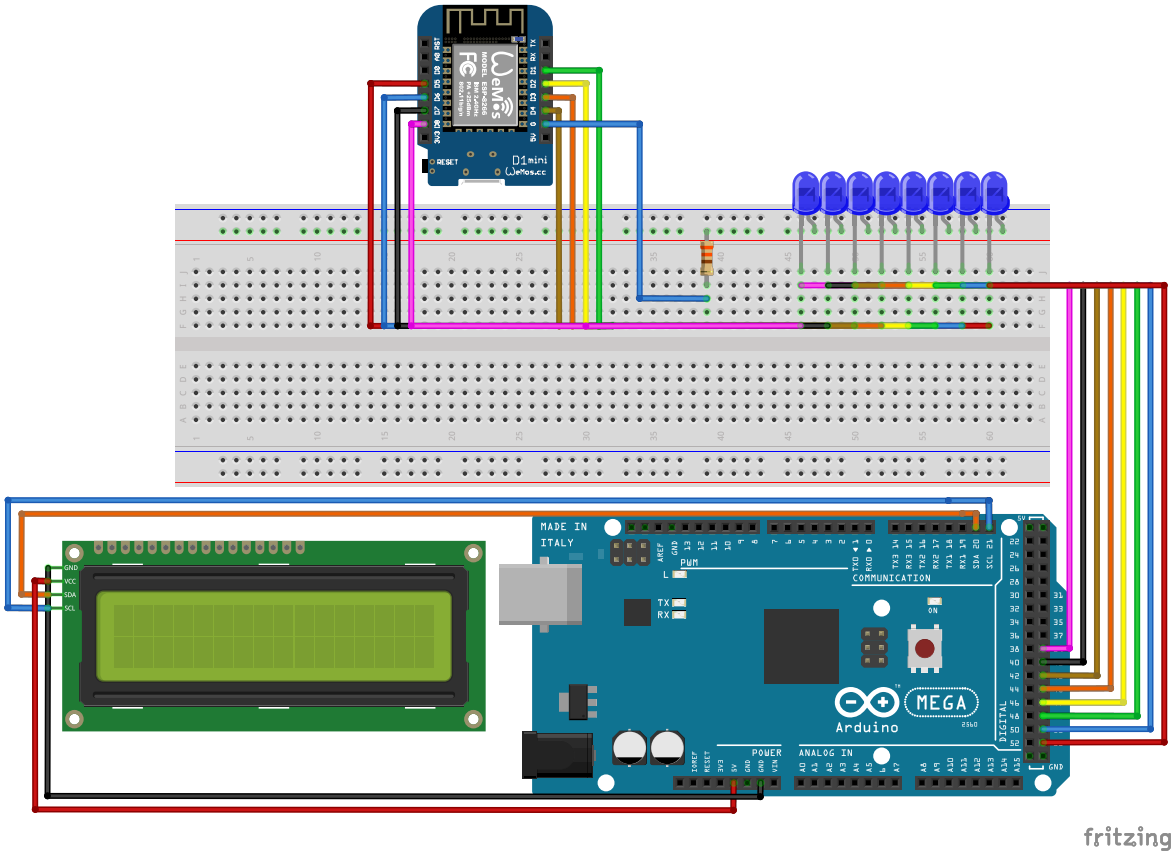
Mithilfe bereitgestellter Materialien erarbeiten die Schüler ein System zur Simulation von binärem Datenaustausch.

- ein Arduino als Sender einer Nachricht (Text), einer als Empfänger
  - Nachrichten werden mit 8 Bit Breite parallel versendet
  - Datenströme sichtbar machen durch LEDs
  - Daten werden durch Empfänger entgegengenommen und auf Display lesbar gemacht
- Umsetzung in einer Projektarbeit über mehrere Stunden
  - je nach Vorwissensstand und Grad der Vorgabe von Code etc. 3-5 Stunden
- Materialien könnten beispielsweise wie eine Softwaredokumentation aufbereitet sein, sodass die Schüler während der Bearbeitung immer wieder auf die Grundlagen zurückgreifen können

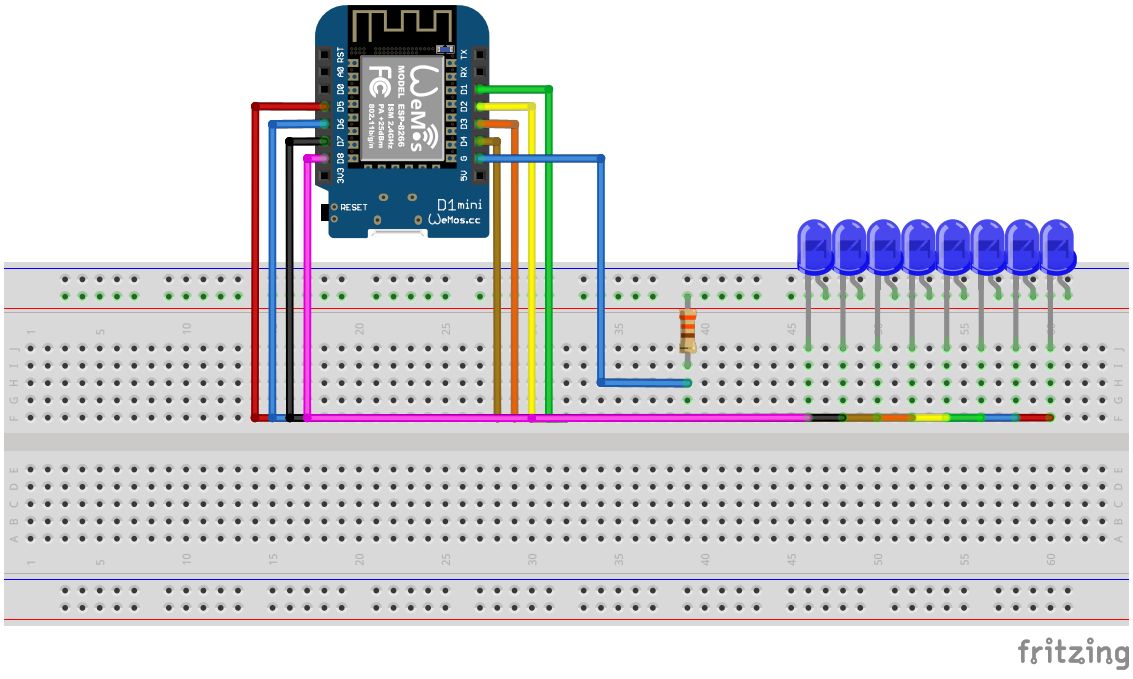
Es folgen nun Code und Schaltpläne, die auch unter `https://github.com/mKobrow/arduino_datenbus` verfügbar sind.

Schaltpläne

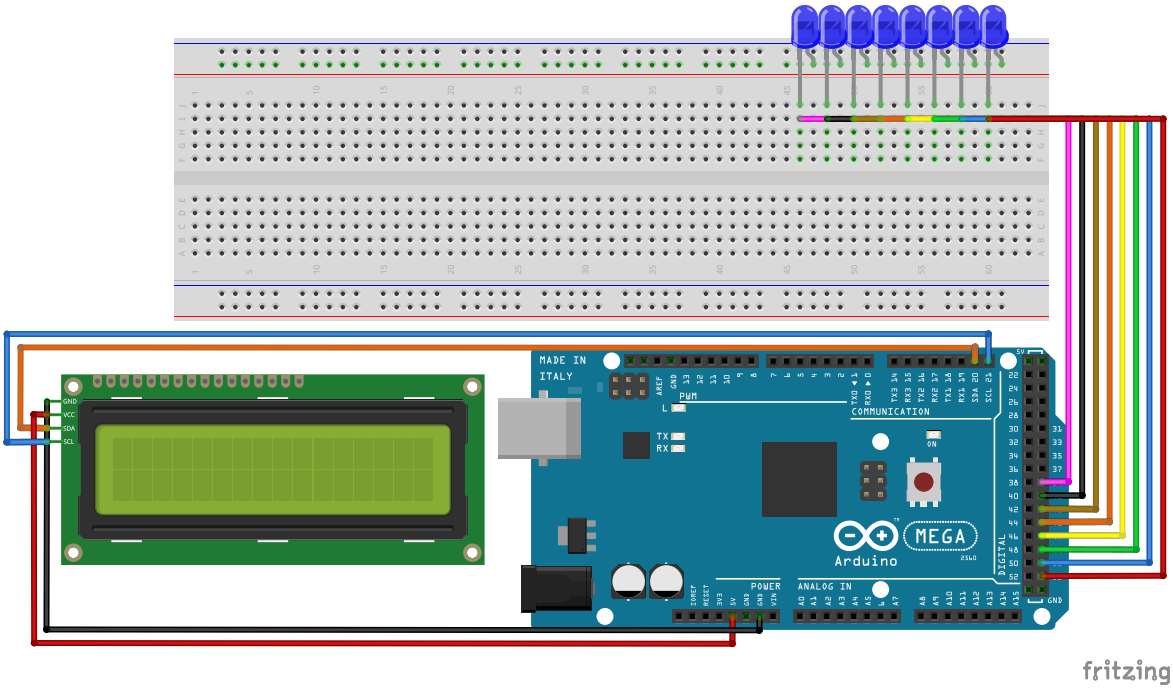
Komplett



Sender



Empfänger


MADE IN
ITALY
PWM
COMMUNICATION
POWER
ANALOG IN
DIGITAL
MEGA
Arduino
fritzing

Code

Sender

```
// Ausgangspins (most significant to least significant)
int dataPins[] = {15,13,2,0,4,5,12,14};

// zu sendene Nachricht (Display ist 32 Zeichen lang)
char message[] = "Hello, World! This may be too long";

// Verzögerung zwischen den gesendeten Nachrichten in ms
int delayTime = 1000;

void setup() {
  // delay für Entprellung des Reset Tasters
  delay(1000);
  // Ausgangspins initialisieren
  for(int i = 0; i < 8; i++) {
    pinMode(dataPins[i],OUTPUT);
  }

  // schreibe STX (start of text)
  for(int i = 0; i < 8; i++) {
    digitalWrite(dataPins[i], bitRead(2, i));
  }

  delay(delayTime);

  // Schleife über Buchstaben der Nachricht
  for(int i = 0; i < sizeof(message); i++) {
    // Schleife über Bits der Buchstaben
    for(int j = 7; j >= 0; j--) {
      digitalWrite(dataPins[j], bitRead(message[i],j));
    }
    delay(delayTime);
  }

  // schreibe ETX (end of text)
  for(int i = 0; i < 8; i++) {
    digitalWrite(dataPins[i], bitRead(3, i));
  }
  delay(1000);
  // alle Pins low damit die LEDs nicht dauerhaft leuchten
  for(int i = 0; i < 8; i++) {
    digitalWrite(dataPins[i], LOW);
  }
}

void loop() {
  // wird nicht benoetigt
}
```

Empfänger

```
// Einbinden der nötigen Bibliotheken für LCD Display und Initialiserung
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Festlegen der Eingangspins (most significant to least significant)
byte dataPins[] = {53,51,49,47,45,43,41,39};

// Verzögerung zwischen den gesendeten Nachrichten in ms
int delayTime = 1000;

byte dataCounter = 0;

void setup() {
  // Initialisierung des Displays
  lcd.init();
  lcd.backlight();
  lcd.setCursor(3,0);
  lcd.print("Waiting...");

  // Einstellen des Pullup Wiederstandes fuer die Eingangspins
  for(int i = 0; i < 8; i++) {
    pinMode(dataPins[i], INPUT_PULLUP);
  }
}

void loop() {
  // lese alle Sekunde ein Zeichen von den Pins
  char character;
  character = readPins();
  delay(delayTime);

  // wenn Zeichen STX (start of text)
  if(character == 2){
    // leere das Display und lese das nächste Zeichen
    character = readPins();
    lcd.clear();
    do {
      // schreibe das Zeichen auf das Display, isPrintable faengt Fehler ab
      if(isPrintable(character)) {
        printLCD(lcd, character, dataCounter);
      }
      // warte, lese naechstes Zeichen
      delay(delayTime);
      dataCounter++;
      character = readPins();
    } while(character != 3);
    dataCounter = 0;
  }
```

```
}

// Funktion zum Lesen eines char von den Eingangspins
char readPins () {
  char c;
  for(int i = 0; i < 8; i++){
    bitWrite(c, i, digitalRead(dataPins[7-i]));
  }
  return c;
}

// schreibt auf LCD und beachtet die maximale LCD laenge
void printLCD (LiquidCrystal_I2C lcd, char c, byte counter) {
  if (counter < 16) {
    lcd.setCursor(counter, 0);
  } else {
    lcd.setCursor(counter-16, 1);
  }

  lcd.print(c);
}
```