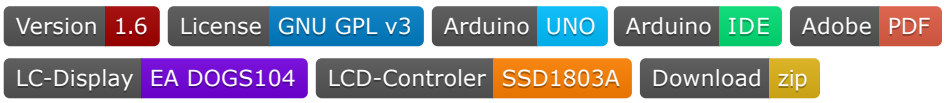
LCD-Shield



Inhaltsverzeichnis

1. Das LCD-Shield

In diesem Repository finden Sie alle Informationen zum LCD-Shield (**L**iquid **C**rystal **D**isplay), welches Sie im Modul "Elektronikdesign" entwickeln. Des Weiteren finden Sie hier eine Bibliothek zum Einbinden in die Arduino IDE (**I**ntegrated **D**evelopment **E**nvironment), um das LC-Display in Betrieb zu nehmen. Die LCD-Shield Bibliothek kann ab der Version 5 verwendet werden. In den folgenden Abschnitten werden die Funktionen, der Stromlaufplan und die Versionsunterschiede des LCD-Shields erklärt.

Funktionen

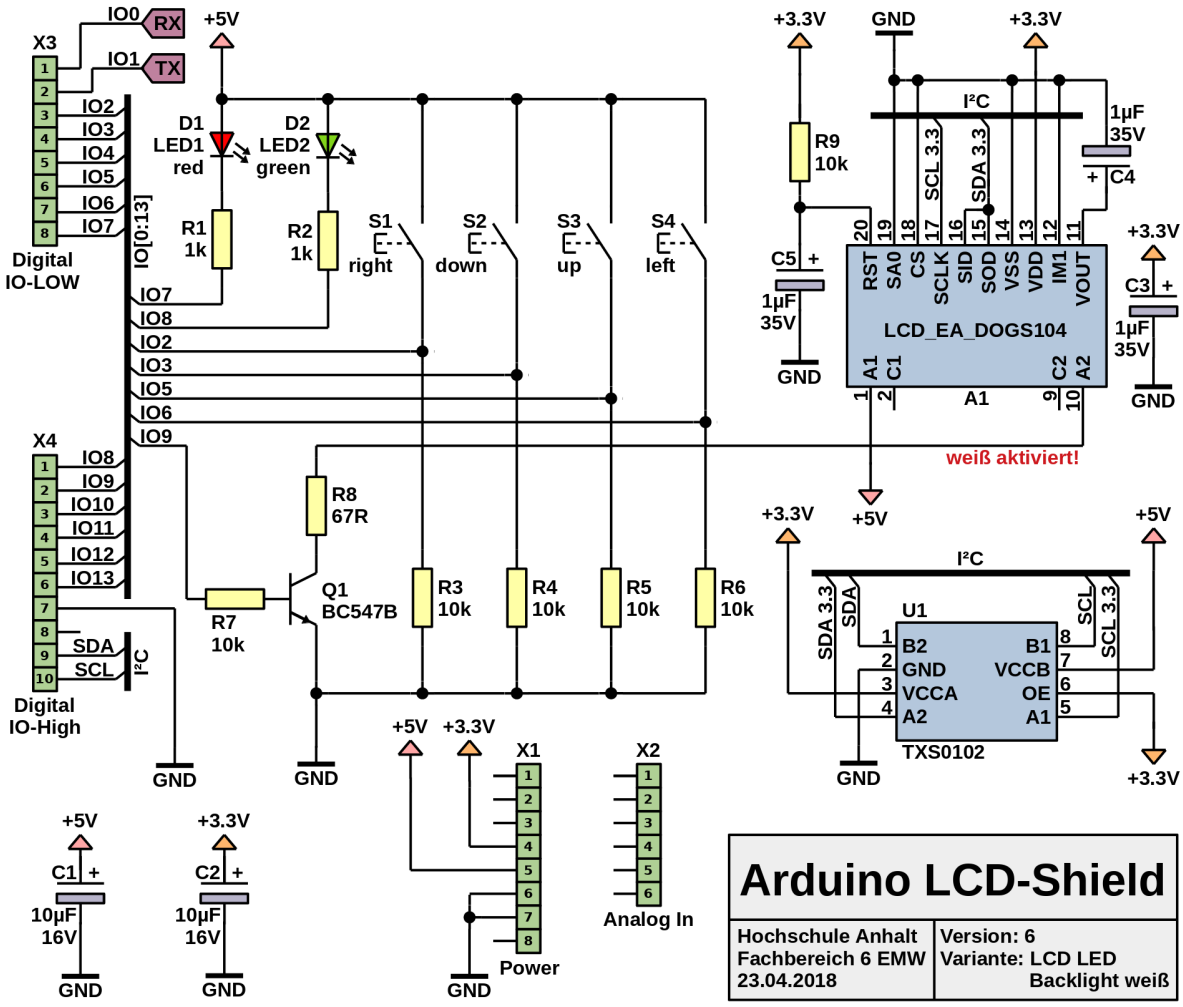
Das LCD-Shield besitzt, neben dem LC-Display, Tasten und LEDs (**L**ight **E**mitting **D**iode), mit denen folgende Funktionen abgebildet werden können:

- Über ein vier mal zehn Zeichen Display können Textnachrichten dargestellt werden.
- Zur Displaynavigation oder zur Realisierung sonstiger Funktionen steht ein Tastenkreuz mit vier Tastern zur Verfügung.

- Mithilfe zweier LEDs (rot und grün) können individuelle Signale dargestellt werden.
- Die Displayhintergrundbeleuchtung lässt sich über Software, mithilfe der bereitgestellten Bibliothek, separat ein- und ausschalten.

Stromlaufplan

Die folgende Abbildung zeigt den Stromlaufplan in der aktuellen Version.



Das LC-Display `A1` mit der Bezeichnung `LCD_EA_DOGS104` besitzt vier Zeilen und kann pro Zeile zehn Zeichen darstellen. Über den Transistor `Q1` kann die Displayhintergrundbeleuchtung ein- und ausgeschaltet werden. Der IC (**I**ntegrated **C**ircuit) `U1` mit der Bezeichnung `TXS0102` ist ein bidirektionaler Pegelwandler für den I²C Bus (**I**nter-**I**ntegrated **C**ircuit). Dieser wird benötigt, da der Arduino mit 5 V arbeitet, aber das LC-Display nur mit maximal 3,3 V angesteuert werden kann. Vier Taster `S1` bis `S4` sind über Pull-Down Widerstände mit dem Arduino verbunden. Die Taster und die LEDs können individuell programmiert werden.

> **Hinweis**: Sollten Sie Fragen zum Stromlaufplan haben, wenden Sie sich bitte an Herrn Prütting.

Versionsunterschiede

In der folgenden Tabelle werden die Versionsunterschiede des LCD-Shields ab Version 5 dargestellt. Dabei werden die Arduino Pins mit den verbundenen Bauelementen verglichen.

Pin	Version 5	Version 6
IO4	S3	-
IO5	S4	S3
IO6	D1	S4
IO7	D2	D1
IO8	Q1	D2
IO9	-	Q1

Bei der Nutzung der LCD-Shield Bibliothek muss entsprechend darauf geachtet werden, welche Shield-Version verwendet wird.

2. LCD-Shield Bibliothek

In den folgenden Abschnitten wird die LCD-Shield Bibliothek erklärt. Dabei wird auf die Installation, die Verwendung von Beispielprogrammen und die Nutzung der Klasse `HSA_LCD_Shield` eingegangen.

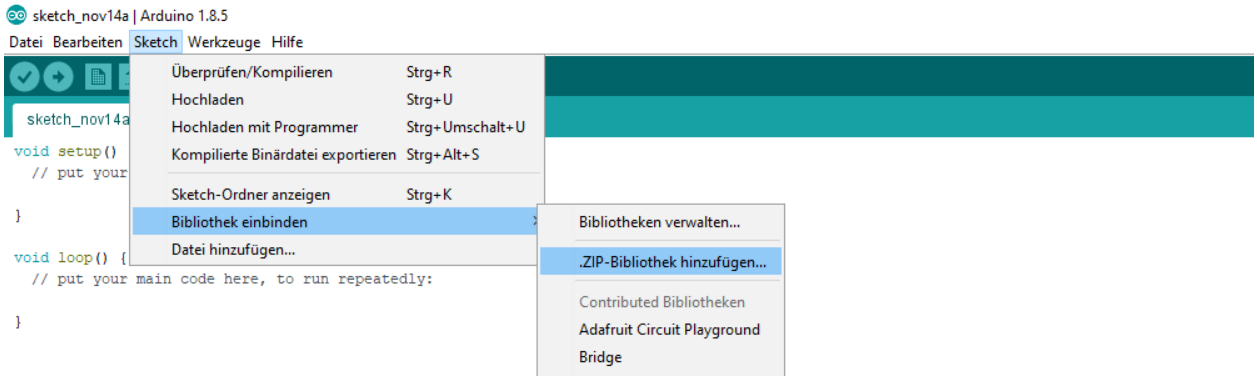
> Sollten zusätzlich Fragen zur Verwendung der LCD-Shield Bibliothek auftauchen, wenden Sie sich bitte an Herrn Müller.

Installation

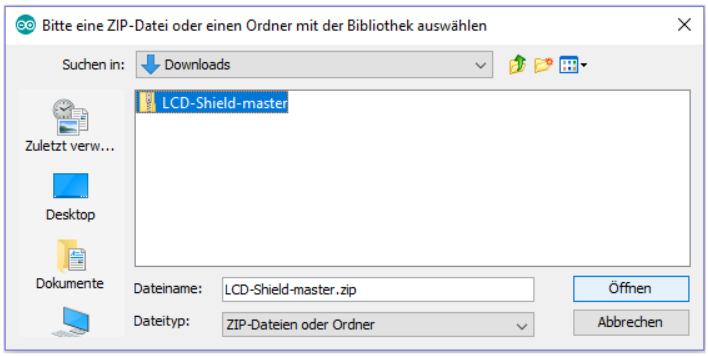
Damit Sie diese Bibliothek in Ihrem Arduino-Sketch verwenden können, laden Sie als Erstes dieses Repository als ZIP-Datei herunter.

Download

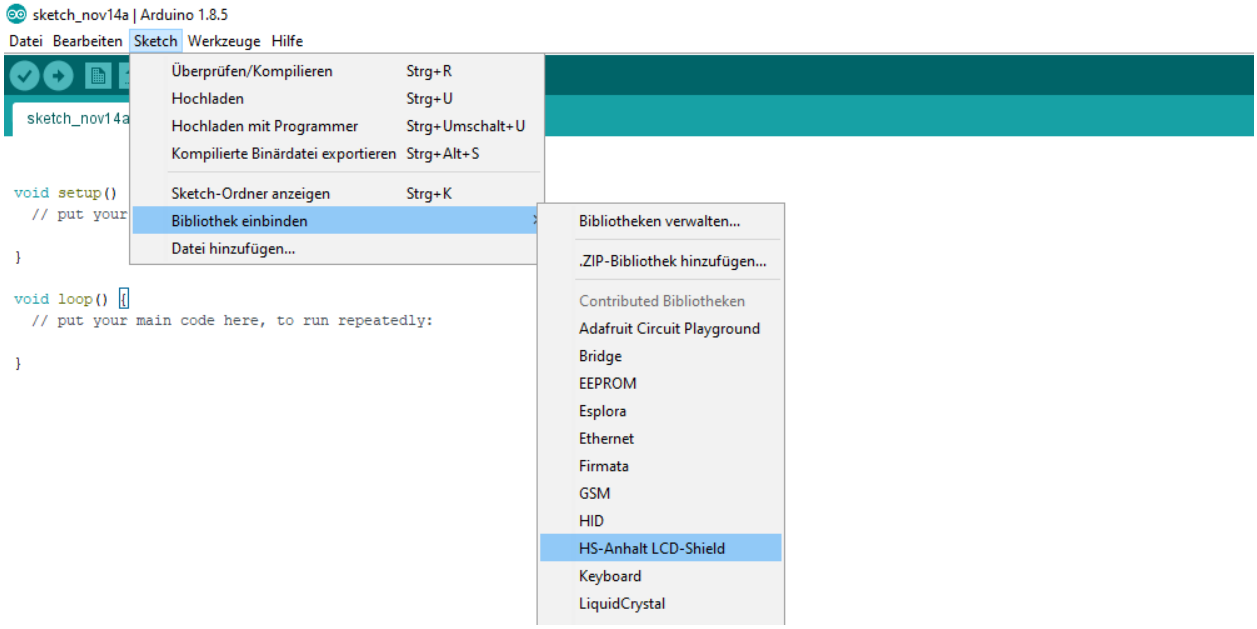
Im Anschluss wechseln Sie zur Arduino IDE und klicken in der Menüleiste auf `Sketch`, `Bibliothek einbinden`, `.ZIP-Bibliothek hinzufügen…`.



Es öffnet sich der Dateimanager, wo Sie die heruntergeladene Datei auswählen und öffnen.



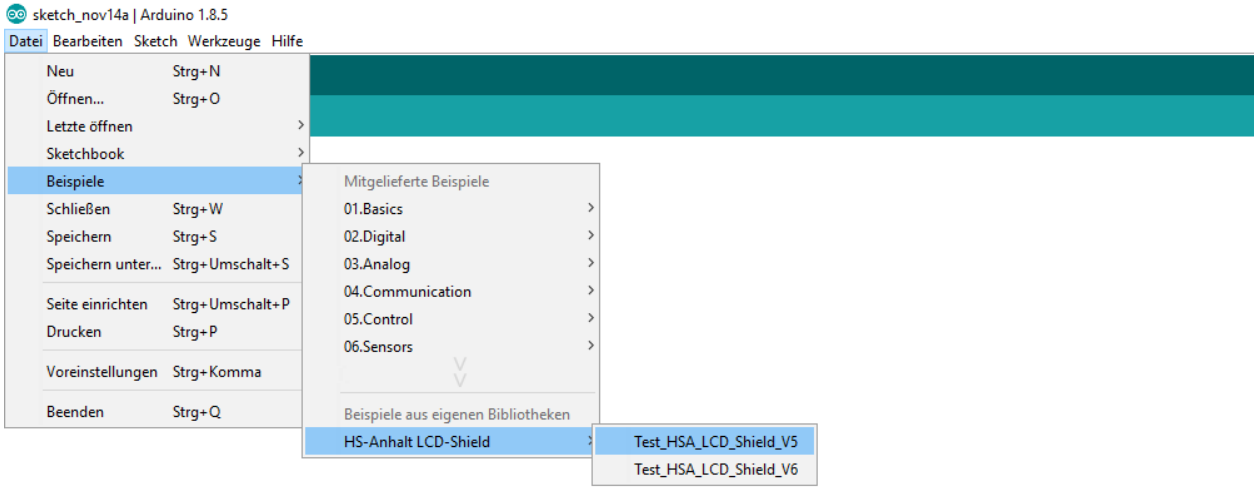
Nun Steht ihnen unter `Sketch`, `Bibliothek einbinden` die `HS-Anhalt LCD-Shield` Bibliothek zur Verfügung und kann in Ihrem Programm eingebunden werden.



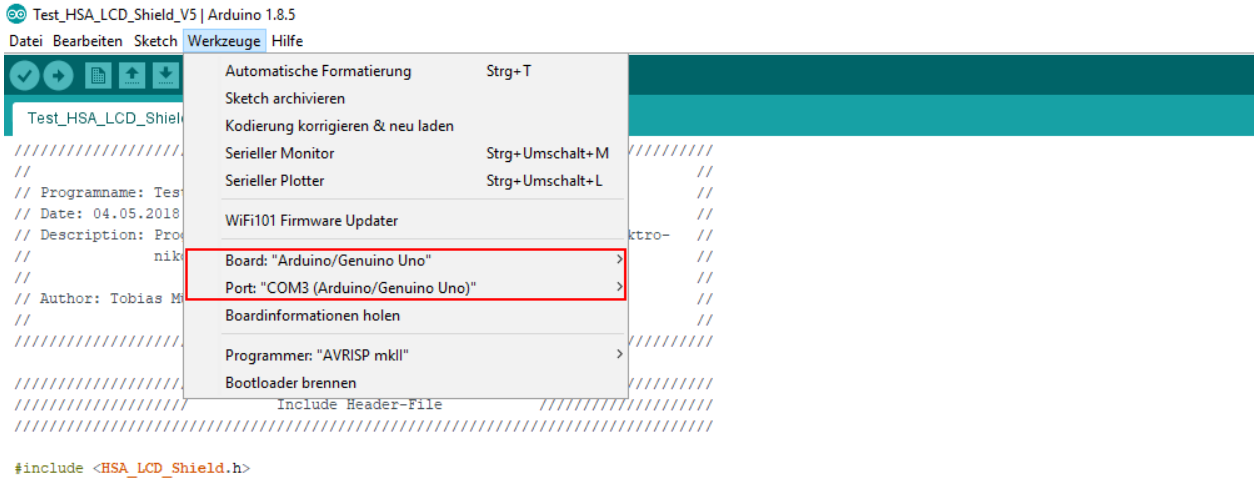
Bespielprogramme

Die Bibliothek beinhaltet Beispielprogramme zum Testen der Funktionalität des LCD-Shields. Des Weiteren dienen die Beispielprogramme zum besseren Verständnis der Klasse `HSA_LCD_Shield`.

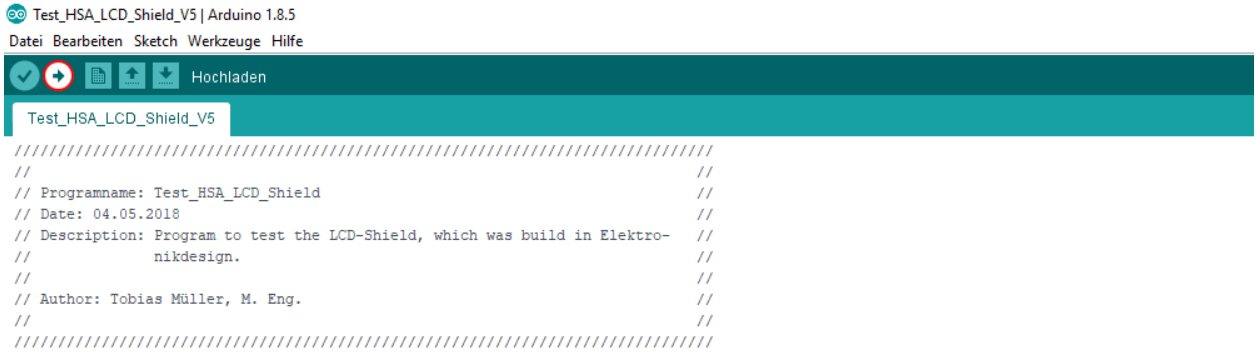
Die Beispielprogramme finden Sie unter `Datei`, `Beispiele`, `HS-Anhalt LCD-Shield`. Entsprechend Ihrer Version des LCD-Shields können Sie das passende Beispielprogramm auswählen.



Kontrollieren Sie unter `Werkzeuge` die Einstellungen: `Board` und `Port`.



Im Anschluss können Sie das Beispielprogramm auf Ihrem Arduino hochladen. Befolgen Sie die Anweisungen auf dem Display, um die LCD-Shield Funktionen zu testen.



Verwenden der Klasse

In den folgenden Abschnitten werden Symbole, Objekte und Methoden der Klasse `HSA_LCD_Shield` erklärt, damit diese im eigenen Programm verwendet werden können. Da die Attribute der Klasse geschützt sind und folglich nicht vom Anwender verändert werden können, werden diese hier nicht weiter aufgeführt oder erläutert.

Symbole

Symbole der Klasse `HSA_LCD_Shield` werden hauptsächlich zur besseren Lesbarkeit des Quellcodes und zur Parameterübergabe für Objekte und Methoden verwendet. In diesem Abschnitt werden alle wichtigen Symbole und deren Werte aufgelistet.

- **Standard LC-Display I²C-Adresse**

```
I2C_ADDRESS         0x3C
```

- **LCD-Shield Versionen**

```
LCD_VERSION_5         0x05
LCD_VERSION_6         0x06
```

- **LCD-Shield Konfigurationen**

```
CONFIG_L1B1         "L1B1"         // Config: LEDs = On, Buttons = On
CONFIG_L0B1         "L0B1"         // Config: LEDs = Off, Buttons = On
CONFIG_L1B0         "L1B0"         // Config: LEDs = On, Buttons = Off
CONFIG_L0B0         "L0B0"         // Config: LEDs = Off, Buttons = Off
```

- **Taster**

```
BUTTON_UP         0x01
BUTTON_RIGHT      0x02
BUTTON_DOWN       0x03
BUTTON_LEFT       0x04
```

- **LEDs**

```
LED_RED         0x05
LED_GREEN       0x06
```

Objekte

Damit Methoden einer Klasse verwendet werden können, muss zunächst ein Objekt der Klasse `HSA_LCD_Shield` erstellt werden. Dafür wird der sogenannte *Konstruktor* einer Klasse aufgerufen. Soll ein Objekt der Klasse wieder entfernt werden, wird der *Destruktor* aufgerufen. Mit Erstellung

eines Objektes werden anhand der übergebenen Parameter Attribute gesetzt, die das Verwenden bestimmter Methoden ermöglicht. Es wird empfohlen Objekte global zu erstellen.

- **Konstruktor:**

  Definition:

  - Standard:

    ```
    HSA_LCD_Shield <objekt>(byte <version>, const char <config>[5]);
    ```

  - Vererbung:

    ```
    HSA_LCD_Shield <Objekt> = HSA_LCD_Shield(byte <version>, const
    char <config>[5]);
    ```

  Beschreibung:

  Erstellt ein `<Objekt>` der Klasse **`HSA_LCD_Shield`**.

  Parameter:

  - `<version>` *(optional)*: Angabe der LCD-Shield Versionsnummer mit `LCD_VERSION_5` oder `LCD_VERSION_6`. Wird kein Parameter übergeben, wird automatisch die höchste Versionsnummer ausgewählt (Standard).
  - `<config>` *(optional)*: Angabe der LED- und Tasterkonfigurationen mit `CONFIG_L1B1`, `CONFIG_L0B1`, `CONFIG_L1B0` und `CONFIG_L0B0`. Hinweise zu den Konfigurationsmöglichkeiten finden Sie unter Symbole. Entsprechend der Konfigutaion werden GPIOs initialisiert. Wird kein Parameter übergeben, werden automatich die Taster und LEDs initialisiert (Standard).

  Beispiel:

  - Beispiel 1:

    ```
    //Einbinden der LCD-Shield Bibliothek
    #include <HSA_LCD_Shield.h>

    // Erstelle Objekt der Klasse mit Standardeinstellungen.
    HSA_LCD_Shield Display;
    ```

  - Beispiel 2:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse für LCD-Shield Version 5 und
Standardeinstellungen
// für LEDs und Taster.
HSA_LCD_Shield Display(LCD_VERSION_5);
```

- Beispiel 3:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse ohne Initialisierung der Taster
und LEDs
// und mit Standardeinstellungen für die LCD-Shield Version.
HSA_LCD_Shield Display(CONFIG_L0B0);
```

- Beispiel 4:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse für LCD-Shield Version 5, ohne
Initialisierung der
// LEDs aber mit Initialisierung der Taster.
HSA_LCD_Shield Display(LCD_VERSION_5, CONFIG_L0B1);
```

- Beispiel 5:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse durch Vererbung mit
Standardeinstellungen.
HSA_LCD_Shield Display = HSA_LCD_Shield();
```

- **Destruktor:**

  Definition:

```
<Objekt>.~HSA_LCD_Shield(void);
```

Beschreibung:

Entfernt ein `<Objekt>` der Klasse `HSA_LCD_Shield`.

Beispiel:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse mit Standardeinstellungen.
HSA_LCD_Shield Display;

 void setup(void) {
    // Führe andere Anweisungen aus!!
    // ...

    // Entferne Objekt der Klasse wieder.
    Display.~HSA_LCD_Shield();
 }

 void loop(void) {

 }
```

Methoden

Methoden sind vom Prinzip her Funktionen einer Klasse. Der Klasse `HSA_LCD_Shield` stehen die Methoden:

- *begin*,
- *returnVersion*,
- *returnConfig*,
- *returnAddress*,
- *lcdBacklight*,
- *controlLED*,
- *getLED*,
- *getButton*,
- *writeRow*,
- *writeXY*
- *clearDisplay*,

zur Verfügung.

- **begin:**

  Definition:

```
bool HSA_LCD_Shield.begin(byte <address>);
```

Beschreibung:

Diese Methode wird als erstes nach der Objekterstellung aufgerufen, um das LC-Display über I^2C zu parametrieren und um, je nach Konfiguration, die GPIOs für LEDs und Taster zu initialisieren. Es wird empfohlen, diese Methode unter der Funktion `void setup(void)` aufzurufen.

Parameter:

`<address>` *(optional)*: Mithilfe dieses Prarameters kann die standard I^2C-Adresse des LC-Displays geändert werden. Wird kein Parameter der Methode übergeben, wird `I2C_ADDRESS` zur Adressierung des LC-Displays verwendet.

Rückgabewert

`bool`:

- `false`: Es ist ein Fehler bei der Konfiguration aufgetreten. Überprüfen Sie die übergebenen Parameter bei der Objekterstellung.
- `true`: Die Konfiguration war erfolgreich.

Beispiel:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse mit Standardeinstellungen.
HSA_LCD_Shield Display;

void setup(void) {
    // Initialisiere serielle Kommunikation mit einer Baudrate von
9600
    Serial.begin(9600);

    // Initialisiere LEDs/Taster und LC-Display mit der standard I²C-
Adresse
    if(Display.begin()) Serial.println("Konfiguration erfolgreich!");
    else Serial.println("Konfiguration nicht erfolgreich!");
}

void loop(void) {

}
```

- **returnVersion:**

Definition:

```
byte HSA_LCD_Shield.returnVersion(void);
```

Beschreibung:

Diese Methode gibt die ausgewählte LCD-Shield Versionsnummer zurück.

Rückgabewert

`byte`: LCD-Shield Versionsnummer

Beispiel:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse mit Standardeinstellungen.
HSA_LCD_Shield Display;

void setup(void) {
    // Initialisiere serielle Kommunikation mit einer Baudrate von
9600
    Serial.begin(9600);

    // Ausgabe der LCD-Shield Versionsnummer
    Serial.print("Es wird die LCD-Shield Version ");
    Serial.print(Display.returnVersion(),DEC);
    Serial.println(" verwendet.");
}

void loop(void) {

}
```

- **returnConfig:**

  Definition:

  ```
  char* HSA_LCD_Shield.returnConfig(void);
  ```

  Beschreibung:

  Diese Methode gibt die ausgewählte Konfiguration für die LEDs und Taster zurück.

  Rückgabewert

`char*`: Speicheradresse zur ausgewählten Konfiguration (String -> Array aus char)

Beispiel:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse mit Standardeinstellungen.
HSA_LCD_Shield Display;

void setup(void) {
    // Initialisiere serielle Kommunikation mit einer Baudrate von
9600
    Serial.begin(9600);

    // Ausgabe der LCD-Shield Versionsnummer
    Serial.print("Es wird der Konfigurationscode \"");
    Serial.print(Display.returnConfig());
    Serial.println("\" verwendet.");
}

void loop(void) {

}
```

- **returnAddress:**

  Definition:

  ```
  byte HSA_LCD_Shield.returnAddress(void);
  ```

  Beschreibung:

  Diese Methode gibt die eingestellte I²C-Adresse des LC-Displays zurück.

  Rückgabewert

  `byte`: I²C-Adresse des LC-Displays

  Beispiel:

  ```
  //Einbinden der LCD-Shield Bibliothek
  #include <HSA_LCD_Shield.h>

  // Erstelle Objekt der Klasse mit Standardeinstellungen.
  HSA_LCD_Shield Display;
  ```

```
void setup(void) {
    // Initialisiere serielle Kommunikation mit einer Baudrate von
9600
    Serial.begin(9600);

    // Ausgabe der LCD-Shield Versionsnummer
    Serial.print("Die eingestellte Adresse des LC-Displays lautet
0x");
    Serial.print(Display.returnAddress(),HEX);
    Serial.println(".");
}

void loop(void) {

}
```

- **lcdBacklight:**

  Definition:

  ```
  bool HSA_LCD_Shield.lcdBacklight(bool <state>, char <brightness>);
  ```

  Beschreibung:

  Mit dieser Methode lässt sich die LCD-Hintergrundbeleuchtung ein- oder ausschalten. Ab Shield-Version 6 lässt sich die Displayhelligkeit einstellen. Methode `HSA_LCD_Shield.begin()` muss erfolgreich ausgeführt worden sein.

  Parameter:

  `<state>`:

  - `HIGH` oder `true`: schaltet die LCD-Hintergrundbeleuchtung ein
  - `LOW` oder `false`: schaltet die LCD-Hintergrundbeleuchtung aus

  `<brightness>` *(optional)*: steuert die Helligkeit der LCD-hintergrungbeleuchtung in Prozent (Wert von `0`-`100`)

  Rückgabewert

  `bool`:

  - `false`: LCD-Backlight nicht konfiguriert
  - `true`: LCD-Backlight erfolgreich angesteuert

  Beispiel:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse mit Standardeinstellungen.
HSA_LCD_Shield Display;

void setup(void) {
    // Initialisiere serielle Kommunikation mit einer Baudrate von
9600
    Serial.begin(9600);

    // Initialisiere LEDs/Taster und LC-Display mit der standard I²C-
Adresse
    Display.begin();

    // LCD-Backlight 6 mal mit 2 Hz blinken lassen
    for(int i = 0x00; i < 0x06; i++) {
        if(Display.lcdBacklight(HIGH))
            Serial.println("LCD-Backlight eingeschaltet.");
        delay(250);
        if(Display.lcdBacklight(LOW))
            Serial.println("LCD-Backlight ausgeschaltet.");
        delay(250);
    }

    // LCD-Backlight dauerhaft einschalten
    if(Display.lcdBacklight(HIGH))
         Serial.println("LCD-Backlight eingeschaltet.");
}

void loop(void) {

}
```

- **controlLED:**

  Definition:

  ```
  bool HSA_LCD_Shield.controlLED(byte <led>, bool <state>);
  ```

  Beschreibung:

  Mit dieser Methode lassen sich die LEDs auf dem LCD-Shield steuern. Die LEDs müssen erfolgreich mit der Methode `HSA_LCD_Shield.begin()` initialisiert worden sein.

  Parameter:

  - `<led>`:

- `LED_RED`: die rote LED wird angesteuert
- `LED_GREEN`: die grüne LED wird angesteuert

- `<state>`:
  - `HIGH` oder `true`: schaltet die LED ein
  - `LOW` oder `false`: schaltet die LED aus

Rückgabewert

`bool`:

- `false`: Es ist ein Fehler bei der LED Ansteuerung aufgetreten. LEDs sind nicht initialisiert.
- `true`: Die LED Ansteuerung war erfolgreich.

Beispiel:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse mit Standardeinstellungen.
HSA_LCD_Shield Display;

void setup(void) {
    // Initialisiere serielle Kommunikation mit einer Baudrate von
9600
    Serial.begin(9600);

    // Initialisiere LEDs/Taster und LC-Display mit der standard I²C-
Adresse
    Display.begin();

    // LED grün und rot 10 mal abwechselnd mit
    // 1 Hz blinken lassen
    for(int i = 0x00; i < 0x0A; i++) {
        if(Display.controlLED(LED_GREEN,HIGH))
        Serial.println("LED gruen eingeschaltet.");
        if(Display.controlLED(LED_RED,LOW))
        Serial.println("LED rot ausgeschaltet.\n");
        delay(500);
        if(Display.controlLED(LED_GREEN,LOW))
        Serial.println("LED gruen ausgeschaltet.");
        if(Display.controlLED(LED_RED,true))
        Serial.println("LED rot eingeschaltet.\n");
        delay(500);
    }
}

void loop(void) {
```

```
}
```

- **getLED:**

  Definition:

  ```
  bool HSA_LCD_Shield.getLED(byte <led>);
  ```

  Beschreibung:

  Mit dieser Methode kann der Status der LEDs auf dem LCD-Shield abgefragt werden. Die LEDs müssen erfolgreich mit der Methode `HSA_LCD_Shield.begin()` initialisiert worden sein.

  Parameter:

  `<led>`:

  - `LED_RED`: die rote LED wird angesteuert
  - `LED_GREEN`: die grüne LED wird angesteuert

  Rückgabewert

  `bool`:

  - `false`: LED ist ausgeschaltet, unbekannt oder nicht initialisiert
  - `true`: LED ist eingeschaltet

  Beispiel:

  ```
  //Einbinden der LCD-Shield Bibliothek
  #include <HSA_LCD_Shield.h>

  // Erstelle Objekt der Klasse mit Standardeinstellungen.
  HSA_LCD_Shield Display;

  void setup(void) {
      // Initialisiere serielle Kommunikation mit einer Baudrate von
  9600
      Serial.begin(9600);

      // Initialisiere LEDs/Taster und LC-Display mit der standard I²C-
  Adresse
      Display.begin();

      // Schalte grüne LED ein
      Display.controlLED(LED_GREEN,HIGH);
  ```

```
    // Frage Status der grünen LED ab
    if(Display.getLED(LED_GREEN))
    Serial.println("gruene LED ist eingeschaltet.");
}

void loop(void) {

}
```

- **getButton:**

  Definition:

  ```
  byte HSA_LCD_Shield.getButton(void);
  ```

  Beschreibung:

  Mit dieser Methode kann der Status der Taster auf dem LCD-Shield abgefragt werden. Die Taster müssen erfolgreich mit der Methode `HSA_LCD_Shield.begin()` initialisiert worden sein.

  Eine Tastenentprellung findet nicht statt.

  Rückgabewert

  `byte`:

  - `0x00` bzw. `false`: kein oder mehr als ein Taster betätigt oder Taster nicht initialisiert
  - `0x01` bzw. `BUTTON_UP`: Taster `UP` betätigt
  - `0x02` bzw. `BUTTON_RIGHT`: Taster `RIGHT` betätigt
  - `0x03` bzw. `BUTTON_DOWN`: Taster `DOWN` betätigt
  - `0x04` bzw. `BUTTON_LEFT`: Taster `LEFT` betätigt

  Beispiel:

  ```
  //Einbinden der LCD-Shield Bibliothek
  #include <HSA_LCD_Shield.h>

  // Erstelle Objekt der Klasse mit Standardeinstellungen.
  HSA_LCD_Shield Display;

  void setup(void) {
      // Initialisiere serielle Kommunikation mit einer Baudrate von
  9600
      Serial.begin(9600);

      // Initialisiere LEDs/Taster und LC-Display mit der standard I²C-
  ```

```
Adresse
    Display.begin();
}

void loop(void) {
    // überwache Tasterzustand
    if(Display.getButton() == BUTTON_UP)
        Serial.println("Taster \'UP\' betaetigt.");
    if(Display.getButton() == BUTTON_RIGHT)
        Serial.println("Taster \'RIGHT\' betaetigt.");
    if(Display.getButton() == BUTTON_DOWN)
        Serial.println("Taster \'DOWN\' betaetigt.");
    if(Display.getButton() == BUTTON_LEFT)
        Serial.println("Taster \'LEFT\' betaetigt.");
    if(Display.getButton() == false)
        Serial.println("Kein Taster betaetigt.");

    // Pause von 500 ms zwischen den Abfragen
    delay(500);
}
```

- **writeRow:**

  Definition:

  Schreibe in alle Zeilen:

  ```
  bool HSA_LCD_Shield.writeRow(const char* <text>);
  ```

  Schreibe in eine Zeile:

  ```
  bool HSA_LCD_Shield.writeRow(byte <row>, const char* <text>);
  ```

  Beschreibung:

  Mit dieser Methode werden Texte (Strings) auf dem LC-Display geschrieben. Es kann dabei gewählt werden, ob in einer gezielten Zeile oder über alle Zeilen hinweg geschrieben werden soll. Dabei werden die entsprechenden Zeilen komplett überschrieben. Das LC-Display muss erfolgreich mit der Methode `HSA_LCD_Shield.begin()` initialisiert worden sein.

  Parameter:

  - `<text>` *(String)*: Textnachricht, die auf dem LC-Display geschrieben werden soll
  - `<row>`:

- `0x01`: Zeile 1 neu schreiben
- `0x02`: Zeile 2 neu schreiben
- `0x03`: Zeile 3 neu schreiben
- `0x04`: Zeile 4 neu schreiben

Rückgabewert

`bool`:

- `false`: LC-Display nicht initialisiert oder Zeilennummer unbekannt
- `true`: Textnachricht erfolgreich auf LC-Display geschrieben

Beispiel:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse mit Standardeinstellungen.
HSA_LCD_Shield Display;

void setup(void) {
    // Initialisiere serielle Kommunikation mit einer Baudrate von
9600
    Serial.begin(9600);

    // Initialisiere LEDs/Taster und LC-Display mit der standard I²C-
Adresse
    Display.begin();

    // LCD-Backlight dauerhaft einschalten
    Display.lcdBacklight(HIGH);

    // "Hallo Welt" auf dem LC-Display schreiben
    if(Display.writeRow("Hallo Welt"))
        Serial.println("Text auf LC-Display geschrieben.");

    // Warte 5s
    delay(5000);

    // "Hallo Welt" in die 3. Zeile auf dem LC-Display schreiben
    if(Display.writeRow(0x03,"Hallo Welt"))
        Serial.println("Text auf LC-Display geschrieben.");
}

void loop(void) {

}
```

- **writeXY:**

Definition:

```
bool HSA_LCD_Shield.writeXY(byte <row>, byte <column>, const char*
<text>);
```

Beschreibung:

Mit dieser Methode werden Texte (Strings) auf dem LC-Display in Abhängigkeit der ausgewählten Zeile/Spalte geschrieben. Das LC-Display muss erfolgreich mit der Methode `HSA_LCD_Shield.begin()` initialisiert worden sein.

Parameter:

- `<text>` *(String)*: Textnachricht, die auf dem LC-Display geschrieben werden soll
- `<row>`: Auswahl der Zeile (`0x01` bis `0x04`)
- `<column>`: Auswahl der Spalte (`0x01` bis `0x0A`)

Rückgabewert

`bool`:

- `false`: LC-Display nicht initialisiert oder Zeilen-/Spaltennummer unbekannt
- `true`: Textnachricht erfolgreich auf LC-Display geschrieben

Beispiel:

```
//Einbinden der LCD-Shield Bibliothek
#include <HSA_LCD_Shield.h>

// Erstelle Objekt der Klasse mit Standardeinstellungen.
HSA_LCD_Shield Display;

void setup(void) {
    // Initialisiere serielle Kommunikation mit einer Baudrate von
9600
    Serial.begin(9600);

    // Initialisiere LEDs/Taster und LC-Display mit der standard I²C-
Adresse
    Display.begin();

    // LCD-Backlight dauerhaft einschalten
    Display.lcdBacklight(HIGH);

    // schreibe "Hallo" auf dem LC-Display in Zeile 1/Spalte 1
    if(Display.writeXY(0x02, 0x01, "Hallo"))
        Serial.println("Text auf LC-Display geschrieben.");
```

```
    // Warte 5s
    delay(5000);

    // schreibe "Hallo" auf dem LC-Display in Zeile 2/Spalte 7
    if(Display.writeXY(0x02, 0x07, "Welt"))
        Serial.println("Text auf LC-Display geschrieben.");
}

void loop(void) {

}
```

- **clearDisplay:**

  Definition:

  ```
  void HSA_LCD_Shield.clearDisplay(void);
  ```

  Beschreibung:

  Diese Methode löscht alle Zeichen auf den LC-Display. Methode `HSA_LCD_Shield.begin()` muss erfolgreich ausgeführt worden sein.

  Rückgabewert

  `bool`:

  - `false`: LC-Display nicht konfiguriert
  - `true`: LC-Display erfolgreich gelöscht

  Beispiel:

  ```
  //Einbinden der LCD-Shield Bibliothek
  #include <HSA_LCD_Shield.h>

  // Erstelle Objekt der Klasse mit Standardeinstellungen.
  HSA_LCD_Shield Display;

  void setup(void) {
      // Initialisiere serielle Kommunikation mit einer Baudrate von
  9600
      Serial.begin(9600);

      // Initialisiere LEDs/Taster und LC-Display mit der standard I²C-
  Adresse
      Display.begin();
  ```

```
    // LCD-Backlight dauerhaft einschalten
    Display.lcdBacklight(HIGH);
}

void loop(void) {
    // "Hallo Welt" auf dem LC-Display mit 1 Hz blinken lassen
    if(Display.writeRow(0x01, "Hallo Welt"))
        Serial.println("Text auf LC-Display geschrieben.");
    delay(500);
    if(Display.clearDisplay())
        Serial.println("Text auf LC-Display geloescht.");
    delay(500);
}
```

3. Literaturverzeichnis

[1] Atmel
ATmega328 (Mikrocontroller) Datenblatt
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
Abfragedatum: 13.11.2018

[2] Texas Instruments
TXS0102 (IC) Datenblatt
http://www.ti.com/lit/ds/symlink/txs0102.pdf
Abfragedatum: 14.11.2018

[3] Arduino
Language Reference
https://www.arduino.cc/en/Reference/HomePage
Abfragedatum: 23.10.2017

[4] SOLOMON SYSTECH
Display-Controller: SSD1803A
https://www.lcd-module.de/fileadmin/eng/pdf/zubehoer/ssd1803a_2_0.pdf
Abfragedatum: 20.04.2018

[5] LCD-Module
LC-Display: EA DOGS104-A
https://www.lcd-module.com/fileadmin/eng/pdf/doma/dogs104e.pdf
Abfragedatum: 20.04.2018

Hochschule Anhalt | Anhalt University of Applied Sciences | Fachbereich 6 EMW
Praktikum Mikrocomputertechnik für EIT, MT und BMT des 3. & 4. Semesters

Prof. Dr.-Ing. Ingo Chmielewski
Ingo.Chmielewski@HS-Anhalt.de

Tobias Müller, M. Eng.
Tobias.Mueller@HS-Anhalt.de

Dipl. Ing. Harald Prütting
Harald.Pruetting@HS-Anhalt.de