

S-Function Tutorial

In diesem Artikel wird erläutert, wie Sie mittels eines S-Function Blocks in Matlab/Simulink funktionstüchtigen Arduino-Code implementieren können. Diese Methode eignet sich hervorragend, um in Simulink Treiber für spezifische Sensoren oder Aktoren zu entwickeln, die dort nicht standardmäßig verfügbar sind. Als anschauliches Beispiel dient die Ansteuerung eines 16x2 LCD-Displays.

Beginnen Sie mit der Implementierung Ihrer Funktion in der Arduino IDE, wobei Sie auf bereits existierende Arduino-Bibliotheken zurückgreifen. Das folgende Bild zeigt, wie ein solches LCD-Display angesteuert wird.

```
Display.ino
1  #include <Wire.h>
2  #include <hd44780.h> // main hd44780 header
3  #include <hd44780ioClass/hd44780_I2Cexp.h> // i2c expander i/o class header
4
5  hd44780_I2Cexp lcd; // declare lcd object: auto locate & auto config expander chip
6
7  // LCD geometry
8  const int LCD_COLS = 16;
9  const int LCD_ROWS = 2;
10
11 void setup()
12 {
13     int status;
14
15     status = lcd.begin(LCD_COLS, LCD_ROWS);
16     if(status) // non zero status means it was unsuccessful
17     {
18         // hd44780 has a fatalError() routine that blinks an led if possible
19         // begin() failed so blink error code using the onboard LED if possible
20         hd44780::fatalError(status); // does not return
21     }
22
23     // initialization was successful, the backlight should be on now
24
25     // Print a message to the LCD
26     lcd.print("Bitte geben Sie");
27     lcd.setCursor(0,1) ;
28     lcd.print("den PIN ein!");
29 }
30
31 void loop()
32 {
33
34 }
35
```

Hierfür wurde die HD44780-Bibliothek von "duinoWitchery" verwendet.

Sobald Sie Ihren Arduino-Code fertiggestellt und dessen Funktionalität überprüft haben, sollten Sie alle verwendeten Bibliotheken herunterladen. Die Arduino-Bibliotheken sind üblicherweise auf

GitHub verfügbar. Für dieses Beispiel wurde die HD44780-Bibliothek von der entsprechenden GitHub-Seite bezogen: [duinoWitchery HD44780 Bibliothek auf GitHub](#).

Zusätzlich kommt in diesem Beispiel die Wire.h-Bibliothek von Arduino zum Einsatz. Diese Bibliothek ist Teil des ArduinoCore AVR GitHub-Projekts und kann unter dem folgenden Link heruntergeladen werden: [ArduinoCore AVR GitHub-Projekt](#). Innerhalb dieses Projekts finden Sie die Wire.h-Datei unter dem Pfad: ArduinoCoreavr/libraries/Wire/.

Im Anschluss erstellen Sie einen Hauptordner für Ihr Simulink-Projekt. Innerhalb dieses Ordners empfiehlt es sich, einen Unterordner für die Bibliotheken anzulegen, den Sie beispielsweise "libs" nennen könnten. In diesen Unterordner speichern Sie alle Bibliotheken, die Sie in Ihrem Arduino-Code verwenden.

Ordner indem ihr Simulink Projekt gespeichert werden soll:

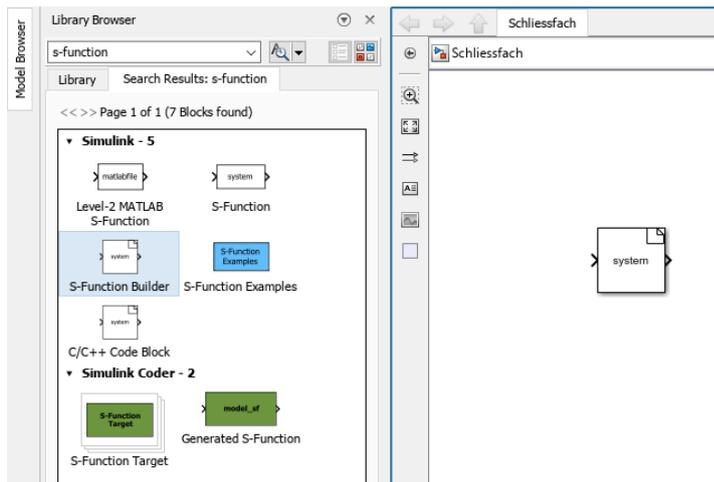
| Name | Date modified | Type | Size |
|------|------------------|-------------|------|
| libs | 17/12/2023 15:35 | File folder | |

Unterordner "libs"

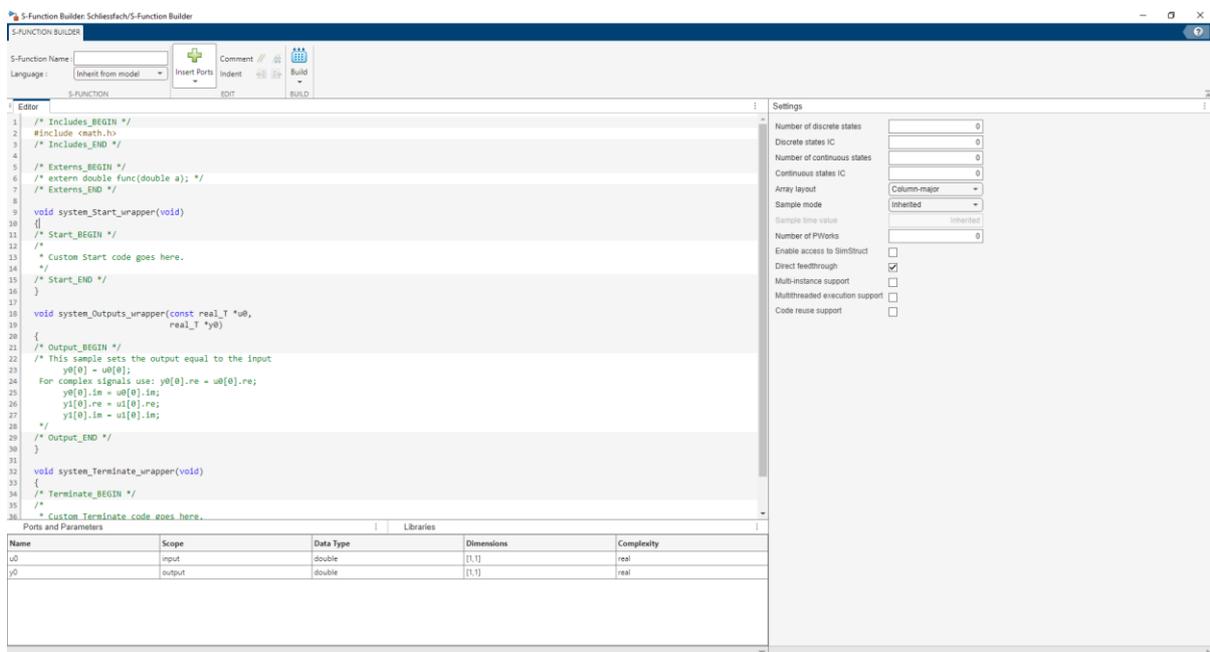
| Name | Date modified | Type | Size |
|-----------------|------------------|-------------|------|
| ArduinoCore-avr | 17/12/2023 15:35 | File folder | |
| hd44780 | 17/12/2023 15:35 | File folder | |

Nachdem Sie die benötigten Bibliotheken heruntergeladen und gespeichert haben, öffnen Sie Matlab/Simulink und erstellen Sie ein neues „Blank Model“. Dieses Modell bildet die Grundlage für Ihr Simulink-Projekt. Sobald das neue Modell in Simulink angelegt ist, wechseln Sie im Matlab zum Projektordner, in dem sich auch der „libs“-Ordner befindet. Anschließend können Sie Ihr Simulink-Projekt speichern.

Als Nächstes öffnen Sie den Library Browser in Simulink und suchen nach dem „S-Function Builder“-Block. Ziehen Sie diesen Block in Ihr Modell.



Um den S-Function Block zu konfigurieren, führen Sie einen Doppelklick auf den Block durch, woraufhin sich ein neues Fenster öffnet, welches folgendermaßen aussieht:



In diesem Fenster nehmen Sie die folgenden Einstellungen vor:

- S-Function Name: Geben Sie hier den Namen der S-Function ein.
- Language: Stellen Sie die Programmiersprache auf C++ ein.
- Number of discrete states: Tragen Sie hier eine 1 ein.
- Sample mode: Stellen Sie den Sample-Modus entsprechend der Funktion des Blocks ein, in diesem Fall auf „Continuous“.

Im zentralen Editor-Block des S-Function Builders können Sie nun Ihren Arduino-Code einfügen. Der Editor-Block unterteilt sich in mehrere Abschnitte, in denen der Code eingetragen wird:

Jeder Codeabschnitt beginnt mit „#ifndef MATLAB_MEX_FILE“ und endet mit „#endif“. Der Include-Teil des Codes beinhaltet die verschiedenen erforderlichen Include-Anweisungen, die aus dem Arduino-Code übernommen werden können. Zusätzlich werden globale Variablen im Include-Teil deklariert.

```

/* Includes_BEGIN */
#ifndef MATLAB_MEX_FILE
#include <Wire.h>
#include <hd44780.h> // main hd44780 header
#include <hd44780ioClass/hd44780_I2Cexp.h> // i2c expander i/o class header

hd44780_I2Cexp lcd; // declare lcd object: auto locate & auto config expander chip

// LCD geometry
const int LCD_COLS = 16;
const int LCD_ROWS = 2;

int displaystate = 1;
#endif

```

Der folgende Abschnitt ist die LCD_Display_Start_wrapper-Funktion. Hier wird die Setup-Funktion aus dem Arduino-Code eingetragen, eingebettet in die „#ifndef MATLAB_MEX_FILE“- und „#endif“-Blöcke.

Das sieht dann so aus:

```

void LCD_Display_Start_wrapper(real_T *xD)
{
/* Start_BEGIN */
#ifndef MATLAB_MEX_FILE
    int status;

    status = lcd.begin(LCD_COLS, LCD_ROWS);
    if(status) // non zero status means it was unsuccessful
    {
        // hd44780 has a fatalError() routine that blinks an led if possible
        // begin() failed so blink error code using the onboard LED if possible
        hd44780::fatalError(status); // does not return
    }

    // initialization was successful, the backlight should be on now

    // Print a message to the LCD
    lcd.print("Bitte geben Sie");
    lcd.setCursor(0,1) ;
    lcd.print("den PIN ein!");

#endif

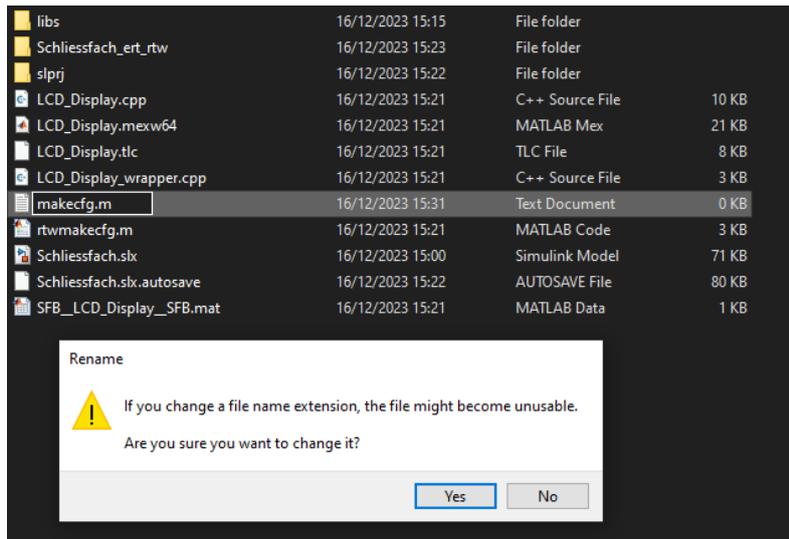
```

Der nächste Schritt in Ihrem Prozess ist die Integration der LCD_Display_Outputs_wrapper-Funktion. Diese Funktion ist normalerweise für die Implementierung der Loop-Funktion aus dem Arduino-Code vorgesehen. In Ihrem Beispiel existiert jedoch keine Loop-Funktion, daher bleibt dieser Abschnitt leer. Es ist wichtig, dass dieser Code wieder innerhalb der „#ifndef MATLAB_MEX_FILE“ und „#endif“ Blöcke eingebettet ist.

Sobald Sie diese Schritte abgeschlossen haben, folgt der Buildprozess für den S-Function Block. Stellen Sie sicher, dass Sie sich im richtigen Ordner in Matlab befinden, bevor Sie auf „Build“ klicken. Im Buildlog sollte eine Bestätigungsnachricht erscheinen, die den erfolgreichen Build signalisiert.

```
### 'LCD_Display.cpp' created successfully
### 'LCD_Display_wrapper.cpp' created successfully
### 'LCD_Display.tlc' created successfully
### S-function 'LCD_Display.mexw64' created successfully
```

Um die Bibliotheken korrekt zu laden, ist es notwendig, eine „makecfg.m“-Datei im Simulink-Ordner zu erstellen. Beginnen Sie damit, eine Textdatei anzulegen und deren Namen entsprechend anzupassen, wobei die Endung von .txt zu .m geändert werden muss. Beim Ändern der Dateiendung erscheint eine Warnmeldung, auf die Sie mit „Ja“ reagieren sollten.



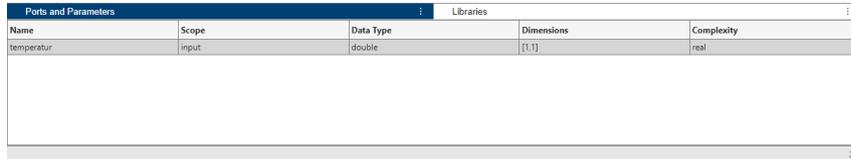
In die makecfg.m-Datei tragen Sie alle verwendeten Bibliotheken ein. Das Schema hierfür sieht folgendermaßen aus:

```
Editor - C:\Users\Danee\Desktop\Schliessfach\makecfg.m
makecfg.m x +
1 function makecfg(objBuildInfo)
2
3     addCompileFlags(objBuildInfo, '-O3');
4
5     AVRWirePath = fullfile('${START_DIR}', 'libs/ArduinoCore-avr/libraries/Wire/src');
6     AVRWirePath2 = fullfile('${START_DIR}', 'libs/ArduinoCore-avr/libraries/Wire/src/utility');
7     HD44780Path = fullfile('${START_DIR}', 'libs/hd44780');
8
9     |
10    addIncludePaths(objBuildInfo, AVRWirePath);
11    addIncludePaths(objBuildInfo, AVRWirePath2);
12    addIncludePaths(objBuildInfo, HD44780Path);
13
14
15    addSourceFiles(objBuildInfo, 'Wire.cpp', AVRWirePath);
16    addSourceFiles(objBuildInfo, 'twi.c', AVRWirePath2);
17    addSourceFiles(objBuildInfo, 'hd44780.cpp', HD44780Path);
18
```

Zuerst benennen Sie einen Pfad für jede Bibliothek, zum Beispiel „AVRWirePath“, der zum Ordner mit den cpp- und h-Dateien führt. Verwenden Sie den Befehl „addIncludePaths“, um den Namen des Pfades anzugeben, und „addSourceFiles“, um den Namen der cpp-Datei und den Pfad der Bibliothek

einzutragen. Führen Sie diesen Schritt für alle verwendeten Bibliotheken durch, speichern Sie die Datei und schließen Sie sie.

Nun können Sie Ihr Simulink-Modell auf den Arduino laden, wodurch die Funktionalität gegeben ist. Möchten Sie beispielsweise die Temperatur eines Sensors auf einem Display ausgeben, so sind einige Anpassungen nötig. Ändern Sie hierfür die Ports und Parameter im S-Function Block:



| Name | Scope | Data Type | Dimensions | Complexity |
|------------|-------|-----------|------------|------------|
| temperatur | input | double | [1,1] | real |

Standardmäßig sind ein „input port“ und ein „output port“ eingestellt. Löschen Sie den Output-Port und benennen Sie den Input-Port, beispielsweise in „Temperatur“, um. Dies wird dann der Name der Variablen. Passen Sie auch den Datentyp an Ihre Bedürfnisse an; für die Temperatur wäre „double“ eine geeignete Wahl.

Hier sehen Sie die LCD_Display_Start_wrapper und LCD_Display_Outputs_wrapper Funktionen dafür:

```
void LCD_Display_Start_wrapper(real_T *xD)
{
  /* Start_BEGIN */
  #ifndef MATLAB_MEX_FILE
    int status;

    status = lcd.begin(LCD_COLS, LCD_ROWS);
    if(status) // non zero status means it was unsuccessful
    {
      // hd44780 has a fatalError() routine that blinks an led if possible
      // begin() failed so blink error code using the onboard LED if possible
      hd44780::fatalError(status); // does not return
    }

    // initialization was successful, the backlight should be on now

    // Print a message to the LCD
    lcd.setCursor(0,0) ;
    lcd.print("Temperatur:");

  #endif
  /* Start_END */
}
```

```

void LCD_Display_Outputs_wrapper(const real_T *temperatur,
                                const real_T *xD)
{
/* Output_BEGIN */
#ifdef MATLAB_MEX_FILE

    lcd.setCursor(0,1);
    lcd.print(temperatur[0]);

#endif
/* Output_END */
}

```

In Ihrem Code werden die Input- und Output-Ports als Arrays betrachtet, die jeweils ein Element enthalten. Daher ist es notwendig, bei der Variablenreferenzierung eckige Klammern mit einer 0 zu setzen. Wiederholen Sie abschließend den Buildprozess, indem Sie erneut auf „Build“ klicken und die alten Dateien überschreiben.

Jetzt können Sie einen Sensor, der die Temperatur misst, an den S-Function Block anschließen und das Simulink-Modell auf den Arduino laden. Das Ergebnis ist die Anzeige der gemessenen Temperatur auf dem Display.