Arduino Uno R4-Treiber für WinFACT

Inhalt

1 Hinweise zur Installation

Zur Installation des Treibers starten Sie das Installationsprogramm *setup.exe*; das Installationsprogramm führt Sie dialoggesteuert durch die Installation. Achten Sie dabei bitte darauf, dass Sie als Zielverzeichnis für den Treiber das Programmverzeichnis Ihrer WinFACT-Installation (bei WinFACT 10 standardmäßig *c:\programme\kahlert\winfact 10*) angeben müssen, damit der Treiber später ordnungsgemäß arbeitet.

Nach der Installation des Treibers finden Sie den Arduino Uno R4-Block in BORIS auf dem Registerblatt *User* der Systemblock-Bibliothek (**Bild 1**).

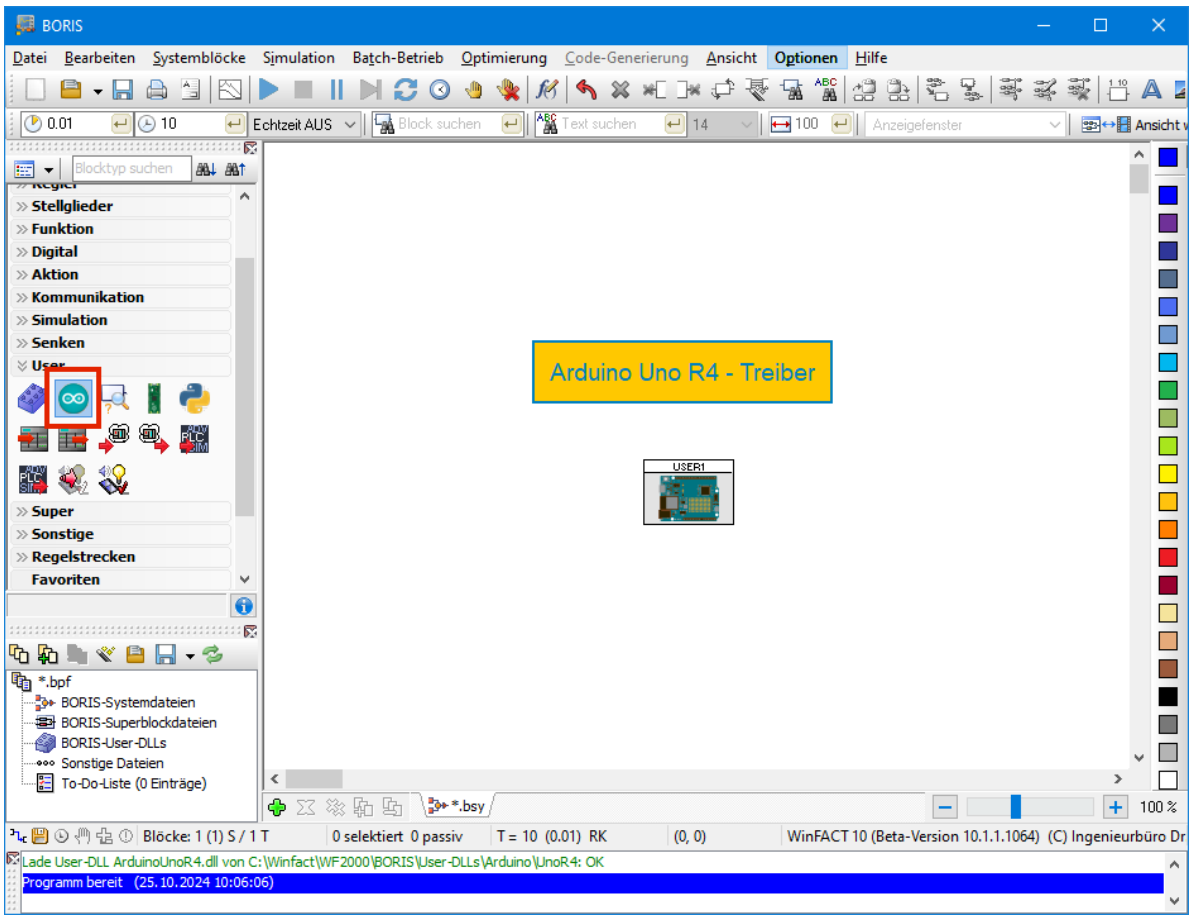


Bild 1 Arduino Uno R4-Block in der BORIS-Systemblock-Bibliothek und auf dem Arbeitsblatt

Durch einen Klick auf das entsprechende Icon der Bibliothek wird ein Arduino Uno R4-Block in die BORIS-Systemstruktur eingefügt. Prinzipiell können beliebig viele Arduino Uno R4-Blöcke (auch mit unterschiedlichen COM-Adressen) in eine BORIS-Simulationsstruktur eingefügt werden. Wie zu erkennen ist, weist der Block nach dem Einfügen zunächst noch keine Ein- oder Ausgänge auf; diese können später nach Belieben hinzugefügt und konfiguriert werden.

Nach Installation des Treibers stehen im Installationsverzeichnis folgende Dateien zur Verfügung:

Dateiname	Funktion
`WFArduinoUnoR4.dll`	BORIS-Treiber-DLL
`WFArduinoUnoR4.ino`	Zugehöriges Arduino-Steuerprogramm
`WFArduinoUnoR4.bmp`	Block-Bitmap

`WFArduinoUnoR4_T.bmp`	Toolbar-Bitmap
`WFArduinoUnoR4_P.bmp`	Drucker-Bitmap
`WFArduinoUnoR4-DigitalIO.bsy`	Beispieldatei Digitale Ein-/Ausgabe
`WFArduinoUnoR4-AnalogIn.bsy`	Beispieldatei Analoge Eingabe
`WFArduinoUnoR4-DACOut.bsy`	Beispieldatei Analoge Ausgabe über DAC
`WFArduinoUnoR4-PWMOut.bsy`	Beispieldatei PWM-Ausgabe
`WFArduinoUnoR4-LEDDigOut.bsy`	Beispieldatei Digitalwertausgabe auf LED-Matrix
`WFArduinoUnoR4-LEDAnaOut.bsy`	Beispieldatei Analogwertausgabe auf LED-Matrix
`WFArduinoUnoR4-TextDigOut.bsy`	Beispieldatei Digitalwertausgabe als Text auf LED-Matrix
`WFArduinoUnoR4-TextAnaOut.bsy`	Beispieldatei Analogwertausgabe als Text auf LED-Matrix
`WFArduinoUnoR4-UserDigIn.bsy`	Beispieldatei Benutzerdefinierte Digitalwerteingabe
`WFArduinoUnoR4-UserAnaIn.bsy`	Beispieldatei Benutzerdefinierte Analogwerteingabe
`WFArduinoUnoR4-UserDigOut.bsy`	Beispieldatei Benutzerdefinierte Digitalwertausgabe
`WFArduinoUnoR4-UserAnaOut.bsy`	Beispieldatei Benutzerdefinierte Analogwertausgabe
`WFArduinoUnoR4-Complete.bsy`	Beispieldatei mit allen Funktionen aus Youtube-Video
`WFArduinoUnoR4.pdf`	dieses Dokument

2 Erforderliche Arduino-Bibliotheken

Für das zum Treiber gehörende Arduino-Steuerprogramm `WFArduinoUnoR4.ino` sind ggfls. folgende Bibliotheken nachzuinstallieren:

```
#include "ArduinoGraphics.h"     // für Textausgabe auf LED-Matrix
#include "Arduino_LED_Matrix.h"  // für Grafikausgabe auf LED-Matrix
#include "RTC.h"                 // für Real-time clock
#include <Servo.h>               // für Beispiel Servo-Ansteuerung
#include <SimpleDHT.h>           // für Beispiel DHT11-Temperatursensor
```

3 Leistungsumfang

Jeder Arduino Uno R4-Block kann bis zu 50 Ein- und Ausgänge aufweisen; zudem können beliebig viele Blöcke in eine BORIS-Simulationsstruktur eingebunden werden. Die an den jeweiligen Block**ein**gängen eingespeisten Signale werden auf dem Arduino weiterverarbeitet, d. h. beispielsweise auf einem GPIO-Pin ausgegeben, auf der LED-Matrix angezeigt oder ähnliches. An den jeweiligen Block**aus**gängen erscheinen Signale, die vom Arduino geliefert werden, also beispielsweise an einem GPIO-Pin eingelesene Spannungen oder über einen am I2C-Bus angeschlossenen Temperatursensor ermittelte Temperaturwerte.

Der Treiber stellt vier unterschiedliche Typen von Ein- bzw. Ausgangssignalen (*Kanaltypen*) zur Verfügung:

- Der Kanaltyp *GPIO* ermöglicht das Einlesen und Ausgeben von digitalen, analogen und PWM-Signalen über die GPIO-Pins des Arduino

- Der Kanaltyp *LED-Matrix* ermöglicht die grafische Darstellung digitaler und analoger Signale auf der LED-Matrix des Arduino
- Der Kanaltyp *Text* ermöglicht die Text-Darstellung digitaler und analoger Signale auf der LED-Matrix des Arduino
- Der Kanaltyp *Benutzerdefiniert* ermöglicht die beliebige Ein- und Ausgabe von Signalen über einen vom Anwender eingefügten Code-Block innerhalb des Arduino-Steuerprogramms. Damit können beispielsweise Werte über den I2C-Bus gelesen oder geschrieben werden, die Echtzeituhr ausgelesen und weiterverarbeitet werden, Werte auf einem externen LC Display ausgegeben werden und ähnliches. Das mitgelieferte Arduino-Steuerprogramm enthält dazu eine Reihe von Beispielen.

4 Parametrierung des Blocks

Bild 2 zeigt den Parameterdialog des Blocks (hier zunächst noch ohne Ein- und Ausgänge).

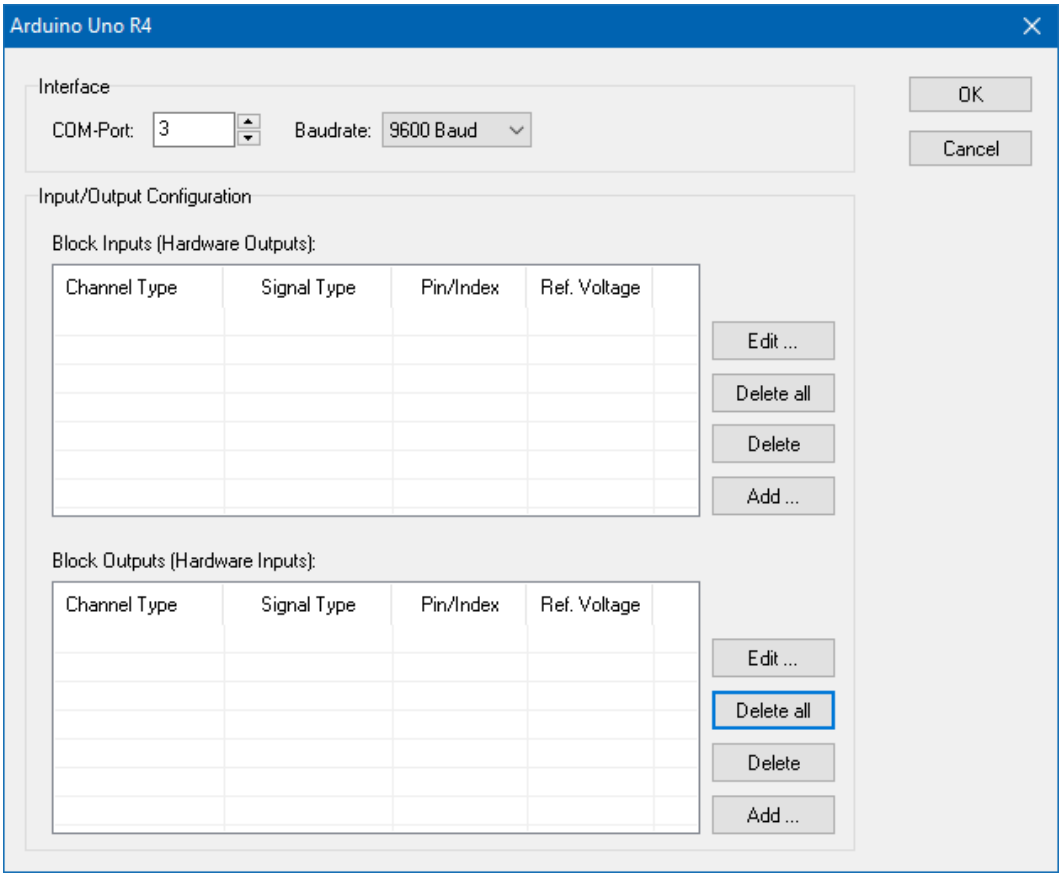


Bild 2 Parameterdialog des (hier noch leeren) Arduino Uno R4-Blocks

Im Gruppenfeld *Interface* können *COM-Port* des Arduino und die gewünschte *Baudrate* eingestellt werden (letztere muss dann im Arduino-Programm ggfls. angepasst werden!). Das Gruppenfeld *Input/Output Configuration* enthält dann die definierten Blockein- und -ausgänge in einer entsprechenden Listenansicht. Über die Schaltflächen rechts der jeweiligen Listenansicht sind folgende Funktionen verfügbar:

Edit …	Aktuell selektierten Ein- bzw. Ausgang bearbeiten
Delete All	Alle aktuellen Ein- bzw. Ausgänge löschen

Delete	Aktuell selektierten Ein- bzw. Ausgang löschen
Add …	Neuen Ein- bzw. Ausgang hinzufügen

Das Bearbeiten eines bereits vorhandenen Ein- oder Ausgangs ist alternativ auch durch Doppelklick auf die entsprechende Zeile in der Listenansicht möglich.

Bild 3 zeigt den Dialog zum Einfügen eines neuen bzw. Bearbeiten eines bereits vorhandenen Ein- oder Ausgangs.

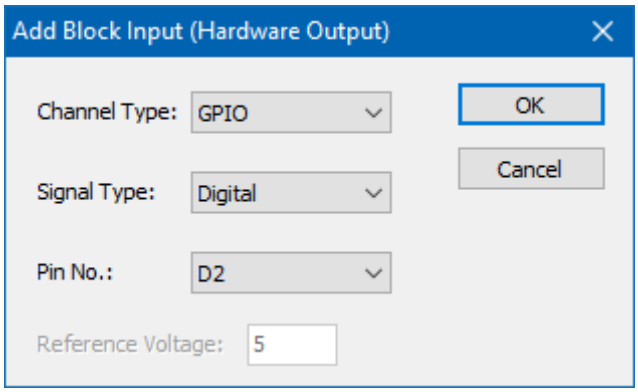


Bild 3 Dialog zum Einfügen eines neuen bzw. Bearbeiten eines vorhandenen Ein- bzw. Ausgangs

Über das Listenfenster *Channel Type* muss zunächst der Kanaltyp (GPIO, LED-Matrix, Text, Benutzerdefiniert) ausgewählt werden. Das Listenfenster *Signal Type* erlaubt dann die Auswahl zwischen einem digitalen, analogen oder PWM-Signal. Das Listenfenster *Pin No*. bzw. *Index No*. legt den verwendeten GPIO-Pin bzw. den gewählten Kanalindex (LED-Matrix, Text, Benutzerdefiniert) fest. Sollen analoge GPIO-Signale verwendet werden, kann über das Editierfeld *Reference Voltage* die zugehörige Bezugsspannung eingegeben werden.

Bild 4 zeigt den Parameterdialog nach Einfügen einiger Kanäle.

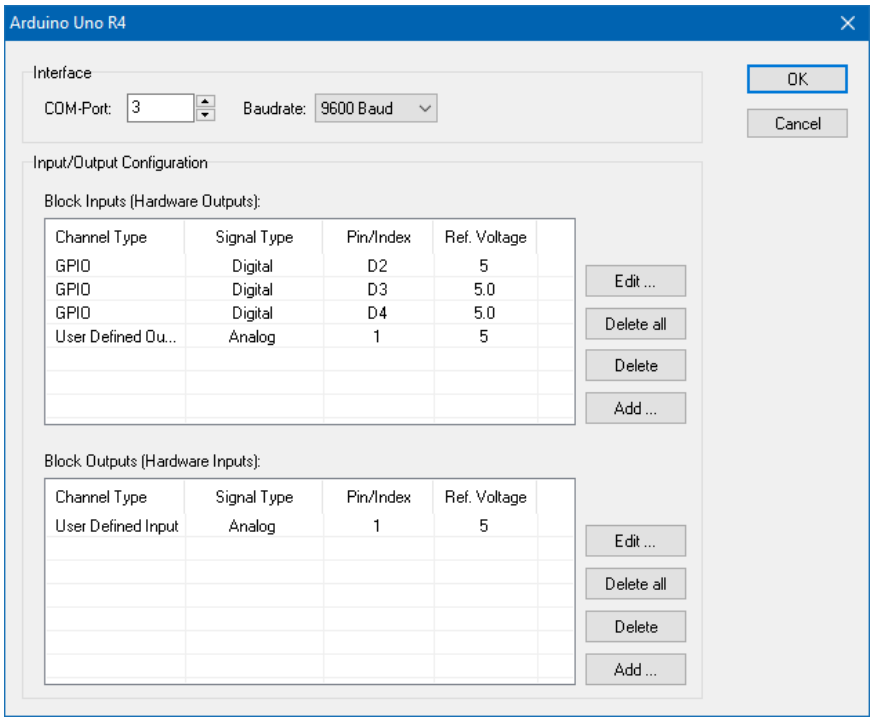


Bild 4 Parameterdialog nach Einfügen einiger Kanäle

5 Block-Ein-/Ausgangssignale

In diesem Abschnitt werden die an den Blockeingängen anzulegenden bzw. an den Blockausgängen ausgegebenen Signale erläutert.

5.1 Block-Eingangssignale

In nachfolgender Tabelle ist s_{in} jeweils der Wert des am Blockeingang angelegten Signals, u_{ref} ist bei analogen Signalen die im Parameterdialog eingestellte Referenzspannung. Die Funktion *round()* rundet auf einen Ganzzahlwert.

Signaltyp	Funktion
GPIO Digital	Für s_{in} > 2.5 wird der GPIO-Pin auf HIGH-Pegel gesetzt, andernfalls auf LOW-Pegel.
GPIO DAC	Der DAC-Ausgang (Pin A0) wird auf den Wert $round\left(\frac{s_{in}}{u_{ref}}4095\right)$ gesetzt.
GPIO PWM	Der PWM-Ausgang wird auf den Wert $round\left(\frac{s_{in}}{u_{ref}}4095\right)$ gesetzt.
LED-Matrix Digital Text Digital Benutzerdef. Digital	Für s_{in} > 2.5 wird der Wert 1 an den Arduino geschickt, andernfalls der Wert 0.
LED-Matrix Analog Text Analog Benutzerdef. Analog	Es wird der Wert $round(s_{in})$ an den Arduino geschickt.

5.2 Block-Ausgangssignale

In nachfolgender Tabelle ist s_{out} jeweils der Wert des am Blockausgang ausgegebenen Signals, u_{ref} ist bei analogen Signalen die im Parameterdialog eingestellte Referenzspannung.

Signaltyp	Funktion
GPIO Digital	Besitzt der GPIO-Pin HIGH-Pegel , wird am Blockausgang der Wert 5, andernfalls der Wert 0 ausgegeben (Standard-Digitalpegel in BORIS).
GPIO Analog	Am Blockausgang wird der Wert $s_{out} = \frac{d}{4095}u_{ref}$ ausgegeben. *d* ist der am GPIO-Pin anliegende Digitalwert.
Benutzerdef. Digital	Liefert die benutzerdefinierte Routine den Wert 1 zurück, wird am Blockausgang der Wert 5, andernfalls der Wert 0 ausgegeben (Standard-Digitalpegel in BORIS).
Benutzerdef. Analog	Der von der benutzerdefinierten Routine zurückgegebene Wert (Ganzzahlwert) wird am Blockausgang ausgegeben.

6 Kommunikationsprotokoll

In diesem Abschnitt wird das vom Treiber verwendete Kommunikationsprotokoll erläutert, d. h. das Format der zwischen BORIS und Arduino ausgetauschten Kommandos bzw. Daten. Für den „normalen“ Einsatz ist dies nicht von Bedeutung, hilfreich ist dieser Abschnitt aber dann, wenn benutzerdefinierte Ein- und Ausgaben (z. B. über Busleitungen) realisiert oder Änderungen beispielsweise an den Routinen zur Unterstützung der LED-Matrix durchgeführt werden sollen.

6.1 Block-Eingänge (Hardware-Ausgabe)

6.1.1 Digitalen GPIO-Pin schreiben

6.1.1.1 Funktion

Setzen eines digitalen GPIO-Pins auf LOW oder HIGH.

6.1.1.2 Kommando BORIS → Arduino

```
!GDx0   // Setzen auf LOW

!GDx1   // Setzen auf HIGH
```

`x` ist die Nummer des zu setzenden Pins (mögliche Werte: 2 … 9)

6.1.1.3 Zugehöriger Arduino-Quellcode

```
pin = orderByte[3] - 48;
pinMode(pin, OUTPUT);
if (orderChar[4] == '0') digitalWrite(pin, LOW);
if (orderChar[4] == '1') digitalWrite(pin, HIGH);
```

6.1.1.4 Beispieldatei

`ArduinoUnoR4-DigitalIO.bsy`

6.1.2 Analogen GPIO-Pin schreiben (DAC)

6.1.2.1 Funktion

Setzen des analogen GPIO-Pins A0 (DAC) auf einen 12-Bit-Wert 0 … 4095

6.1.2.2 Kommando BORIS → Arduino

```
!GAyyyy
```

`yyyy` ist der auszugebende Analogwert (`0000` … `4095`)

6.1.2.3 Zugehöriger Arduino-Quellcode

```
int make12BitValue(byte t, byte h, byte z, byte e, int maxValue)
{
  int g = (e - 48) + (z - 48)*10 + (h - 48)*100 + (t - 48)*1000;
  if (g < 0) g = 0;
```

```
  if (g > maxValue) g = maxValue;
  return g;
}

analogWriteResolution(12);
analogWrite(A0, make12BitValue(orderByte[3], orderByte[4], orderByte[5],
                               orderByte[6], 4095));
```

6.1.2.4 Beispieldatei

`ArduinoUnoR4-DACOut.bsy`

6.1.3 PWM-GPIO-Pin schreiben

6.1.3.1 Funktion

Setzen eines als PWM-Ausgang konfigurierten GPIO-Pins auf einen 12-Bit-Wert 0 … 4095

6.1.3.2 Kommando BORIS → Arduino

```
!GPxyyyy
```

`x` ist die Nummer des zu setzenden Pins (mögliche Werte: 3, 5, 6, 9)

`yyyy` ist der auszugebende Analogwert (`0000` … `4095`)

6.1.3.3 Zugehöriger Arduino-Quellcode

```
int make12BitValue(byte t, byte h, byte z, byte e, int maxValue)
{
  int g = (e - 48) + (z - 48)*10 + (h - 48)*100 + (t - 48)*1000;
  if (g < 0) g = 0;
  if (g > maxValue) g = maxValue;
  return g;
}

pin = orderByte[3] - 48;
pinMode(pin, OUTPUT);
analogWriteResolution(12);
analogWrite(pin, make12BitValue(orderByte[4], orderByte[5], orderByte[6],
                                orderByte[7], 4095));
```

6.1.3.4 Beispieldatei

`ArduinoUnoR4-PWMOut.bsy`

6.1.4 Digitalwert auf LED-Matrix ausgeben

6.1.4.1 Funktion

Gibt einen Digitalwert (0 oder 1) als entsprechendes Symbol auf der LED-Matrix aus.

6.1.4.2 Kommando BORIS → Arduino

```
!MDx0    // Wert 0 ausgeben

!MDx1    // Wert 1 ausgeben
```

`x` ist der Index des auszugebenden Kanals (wird hier nicht benutzt)

6.1.4.3 Zugehöriger Arduino-Quellcode

```
index = orderByte[3] - 48;  // wird in diesem Beispiel nicht genutzt!
if (orderChar[4] == '0') matrix.renderBitmap(frame0, 8, 12);  // '0' ausgeben
if (orderChar[4] == '1') matrix.renderBitmap(frame1, 8, 12);  // '1' ausgeben
```

Die beiden verwendeten LED-Matrix-Bitmaps `frame0` und `frame1` sind am Anfang des Arduino-Steuerprogramms deklariert.

6.1.4.4 Beispieldatei

`ArduinoUnoR4-LEDDigOut.bsy`

6.1.5 Analogwert auf LED-Matrix ausgeben

6.1.5.1 Funktion

Gibt einen Analogwert als Säulengrafik auf der LED-Matrix aus

6.1.5.2 Kommando BORIS → Arduino

```
!MDxyyyy
```

`x` ist der Index des auszugebenden Kanals (hier die gewählte Spalte der LED-Matrix, mögliche Werte `1` … `9`)

`yyyy` ist der auszugebende Analogwert (`0000` … `9999`), hier sind nur Werte zwischen 0 und 8 sinnvoll!

6.1.5.3 Zugehöriger Arduino-Quellcode

```
// Ausgabe des Analogwerts (0 .. 8) als Säule in der durch index festgelegten
// Spalte der LED-Matrix
index = orderByte[3] - 48;
value = (orderByte[7] - 48) + (orderByte[6] - 48)*10 + (orderByte[5] - 48)*100
        + (orderByte[4] - 48)*1000;
if (value < 0) value = 0;
if (value > 8) value = 8;
for (int i=0; i<value; i++) frame[7-i][index-1] = 1;
for (int i=value; i<8; i++) frame[7-i][index-1] = 0;
matrix.renderBitmap(frame, 8, 12);
```

Das verwendete LED-Matrix-Bitmap `frame` ist am Anfang des Arduino-Steuerprogramms deklariert.

6.1.5.4 Beispieldatei

`ArduinoUnoR4-LEDAnaOut.bsy`

6.1.6 Digitalwert als Text auf LED-Matrix ausgeben

6.1.6.1 Funktion

Gibt einen Digitalwert (0 oder 1) als Text auf der LED-Matrix aus

6.1.6.2 Kommando BORIS → Arduino

```
!TDx0   // Wert 0 ausgeben

!TDx1   // Wert 1 ausgeben
```

`x` ist der Index des auszugebenden Kanals (legt hier horizontale Text-Startposition fest, mögliche Werte `0` … `9`, sinnvolle Werte hier `1` … `3`)

6.1.6.3 Zugehöriger Arduino-Quellcode

```
index = orderByte[3] - 48;  // Index (legt in diesem Beispiel
                            // Startposition der Ausgabe fest)
matrix.beginDraw();
matrix.stroke(0xFFFFFFFF);
matrix.textFont(Font_4x6);
if (orderChar[4] == '0') matrix.text("0", (orderByte[3]-48-1)*4, 2);
if (orderChar[4] == '1') matrix.text("1", (orderByte[3]-48-1)*4, 2);
matrix.endDraw();
```

6.1.6.4 Beispieldatei

`ArduinoUnoR4-TextDigOut.bsy`

6.1.7 Analogwert als Text auf LED-Matrix ausgeben

6.1.7.1 Funktion

Gibt einen Analogwert als Text auf der LED-Matrix aus

6.1.7.2 Kommando BORIS → Arduino

```
!TAxyyy
```

`x` ist der Index des auszugebenden Kanals (wird hier nicht benutzt)

`yyy` ist der auszugebende Analogwert (`000` … `999`)

6.1.7.3 Zugehöriger Arduino-Quellcode

```
index = orderByte[3] - 48;  // Index (wird hier ignoriert!)
matrix.beginDraw();
matrix.stroke(0xFFFFFFFF);
matrix.textFont(Font_4x6);
outText[0] = orderChar[4];
```

```
outText[1] = orderChar[5];
outText[2] = orderChar[6];
matrix.text(outText, 0, 2);
matrix.endDraw();
```

6.1.7.4 Beispieldatei

`ArduinoUnoR4-TextAnaOut.bsy`

6.1.8 Benutzerdefinierte Digitalwert-Ausgabe

6.1.8.1 Funktion

Gibt einen Digitalwert in einer benutzerdefinierten Routine aus.

6.1.8.2 Kommando BORIS → Arduino

```
!UDx0   // Wert 0 ausgeben

!UDx1   // Wert 1 ausgeben
```

`x` ist der in BORIS gewählte Index (mögliche Werte: `0` … `9`)

Durch Nutzung unterschiedlicher Indizes ist es möglich, bis zu 10 verschiedene benutzerdefinierte Ausgaberoutinen zur erstellen, die dann in Abhängigkeit vom gewählten Index ausgeführt werden.

6.1.8.3 Zugehöriger Arduino-Quellcode

In diesem Beispiel-Code wird die Board-LED ein- bzw. ausgeschaltet.

```
// Beispiel: Board-LED ein/aus
index = orderByte[3] - 48;  // Index (in diesem Beispiel
                            //        nicht benutzt)
if (orderChar[4] == '0') digitalWrite(LED_BUILTIN, LOW);
if (orderChar[4] == '1') digitalWrite(LED_BUILTIN, HIGH);
```

6.1.8.4 Beispieldatei

`ArduinoUnoR4-UserDigOut.bsy`

6.1.9 Benutzerdefinierte Analogwert-Ausgabe

6.1.9.1 Funktion

Gibt einen Analogwert in einer benutzerdefinierten Routine aus.

6.1.9.2 Kommando BORIS → Arduino

```
!UAxyyy
```

`x` ist der in BORIS gewählte Index (mögliche Werte: `0` … `9`)

`yyyy` ist der auszugebende Analogwert (`0000` … `9999`)

Durch Nutzung unterschiedlicher Indizes ist es möglich, bis zu 10 verschiedene benutzerdefinierte Ausgaberoutinen zur erstellen, die dann in Abhängigkeit vom gewählten Index ausgeführt werden.

6.1.9.3 Zugehöriger Arduino-Quellcode

In diesem Beispiel-Code wird ein kleiner Servo-Motor[1] angesteuert (Schaltplan im Anhang!). Die Sollposition des Motors (0 … 180 °) wird über `yyyy` vorgegeben.

```
#include <Servo.h>  // für Servo-Ansteuerung

int pinServo = 10;  // Pin für Data-Anschluss des Servo
Servo servo;        // Servo-Objekt

int make12BitValue(byte t, byte h, byte z, byte e, int maxValue)
{
  int g = (e - 48) + (z - 48)*10 + (h - 48)*100 + (t - 48)*1000;
  if (g < 0) g = 0;
  if (g > maxValue) g = maxValue;
  return g;
}

// Beispiel: Servo-Ansteuerung über Pin 10
index = orderByte[3] - 48;  // Index (hier nicht benutzt)
servo.write(make12BitValue(orderByte[4], orderByte[5], orderByte[6],
                           orderByte[7], 9999));
```

6.1.9.4 Beispieldatei

`ArduinoUnoR4-UserAnaOut.bsy`

6.2 Block-Ausgänge (Hardware-Eingabe)

6.2.1 Digitalen GPIO-Pin lesen

6.2.1.1 Funktion

Statusabfrage eines digitalen GPIO-Pins.

6.2.1.2 Kommando BORIS → Arduino

```
?GDx
```

`x` ist die Nummer des abzufragenden Pins (mögliche Werte: 2 … 9)

[1] Einen solchen Motor findet man in vielen Arduino Sensor Kits, z. B. unter www.elegoo.com oder https://funduino.de/

6.2.1.3 Antwort Arduino → BORIS [2]

```
0 (LOW) oder 1 (HIGH)
```

6.2.1.4 Zugehöriger Arduino-Quellcode

```
pin = orderByte[3] - 48;
pinMode(pin, INPUT);
int wert = digitalRead(pin);
if (wert == LOW)
  Serial.println('0');
if (wert == HIGH)
  Serial.println('1');
```

6.2.1.5 Beispieldatei

`ArduinoUnoR4-DigitalIO.bsy`

6.2.2 Analogen GPIO-Pin lesen

6.2.2.1 Funktion

Wertabfrage eines analogen GPIO-Pins.

6.2.2.2 Kommando BORIS → Arduino

```
?GAx
```

`x` ist die Nummer des abzufragenden Pins (mögliche Werte: 0 … 5)

6.2.2.3 Antwort Arduino → BORIS

```
0 … 4095
```

6.2.2.4 Zugehöriger Arduino-Quellcode

```
pin = orderByte[3] - 48 + 14; // A0 hat Pinnummer 14, A1 Pinnummer 15 usw.!
analogReadResolution(12);
int wert = analogRead(pin);
Serial.println(wert);
```

6.2.2.5 Beispieldatei

`ArduinoUnoR4-AnalogIn.bsy`

[2] Antworten des Arduino müssen immer mit <CR> <LF> abgeschlossen werden, also z. B. über `Serial.println` ausgegeben werden!

6.2.3 Benutzerdefinierte Digitalwert-Eingabe

6.2.3.1 Funktion

Liest einen Digitalwert in einer benutzerdefinierten Routine aus.

6.2.3.2 Kommando BORIS → Arduino

```
?UDx
```

`x` ist der in BORIS gewählte Index (mögliche Werte: `0` … `9`)

Durch Nutzung unterschiedlicher Indizes ist es möglich, bis zu 10 verschiedene benutzerdefinierte Eingaberoutinen zur erstellen, die dann in Abhängigkeit vom gewählten Index ausgeführt werden.

6.2.3.3 Antwort Arduino → BORIS

```
0 oder 1
```

6.2.3.4 Zugehöriger Arduino-Quellcode

In diesem Beispiel werden Temperatur und Luftfeuchtigkeit über einen Sensor vom Typ DHT11[3] eingelesen (Schaltplan im Anhang!). Liegt die Temperatur über 20 °C, so wird 1-Signal (HIGH) zurückgegeben, andernfalls 0-Signal (LOW).

```
#include <SimpleDHT.h>           // für DHT11-Temperatursensor

// DHT11-Temperatursensor
int pinDHT11 = 11;           // Pin für Data-Anschluss des DHT11
SimpleDHT11 dht11(pinDHT11); // DHT11-Objekt
byte temperature = 0;
byte humidity = 0;

// Beispiel: Überprüfung, ob Temperatur > 20 °C (DHT11-Sensor)
index = orderByte[3] - 48;  // Index (hier nicht benutzt)
byte newTemperature;
byte newHumidity;
if (dht11.read(&newTemperature, &newHumidity, NULL) == SimpleDHTErrSuccess)
{
  temperature = newTemperature;
  humidity = newHumidity;
}
if (temperature > 20) Serial.println('1'); else Serial.println('0');
delay(500);
```

6.2.3.5 Beispieldatei

`ArduinoUnoR4-UserDigIn.bsy`

[3] Einen solchen Sensor findet man in vielen Arduino Sensor Kits, z. B. unter www.elegoo.com oder https://funduino.de/

6.2.4 Benutzerdefinierte Analogwert-Eingabe

6.2.4.1 Funktion

Liest einen Analogwert in einer benutzerdefinierten Routine aus.

6.2.4.2 Kommando BORIS → Arduino

```
?UAx
```

`x` ist der in BORIS gewählte Index (mögliche Werte: `0` … `9`)

Durch Nutzung unterschiedlicher Indizes ist es möglich, bis zu 10 verschiedene benutzerdefinierte Eingaberoutinen zur erstellen, die dann in Abhängigkeit vom gewählten Index ausgeführt werden.

6.2.4.3 Antwort Arduino → BORIS

```
0 … yyy…yyy
```

Die Antwort des Arduino muss ein Ganzzahlwert sein, kann aber beliebig viele Stellen besitzen.

6.2.4.4 Zugehöriger Arduino-Quellcode

In diesem Beispiel werden je nach gewähltem Index die aktuelle Stunden-, Minuten- oder Sekundenzahl der Echtzeituhr bzw. Temperatur und Luftfeuchtigkeit über einen Sensor vom Typ DHT11[4] eingelesen (Schaltplan im Anhang!). Temperatur und Luftfeuchtigkeit werden in einem gemeinsamen Ganzzahlwert an BORIS übergeben und dort wieder in die beiden Einzelwerte aufgesplittet.

```
#include "RTC.h"                 // für Real-time clock
#include <SimpleDHT.h>           // für DHT11-Temperatursensor

// DHT11-Temperatursensor
int pinDHT11 = 11;           // Pin für Data-Anschluss des DHT11
SimpleDHT11 dht11(pinDHT11); // DHT11-Objekt
byte temperature = 0;
byte humidity = 0;

// Real-time clock RTC
RTCTime currentTime;
RTC.begin();         // RTC initialisieren
RTCTime startTime(30, Month::JUNE, 2023, 00, 00, 00,
                  DayOfWeek::WEDNESDAY, SaveLight::SAVING_TIME_ACTIVE
RTC.setTime(startTime); // RTC starten

// Beispiel: Abfrage von RTC, Temperatur oder Luftfeuchtigkeit (DHT11-Sensor)
index = orderByte[3] - 48;  // Index (wählt hier zwischen Stunden, Minuten
```

[4] Einen solchen Sensor findet man in vielen Arduino Sensor Kits, z. B. unter www.elegoo.com oder https://funduino.de/

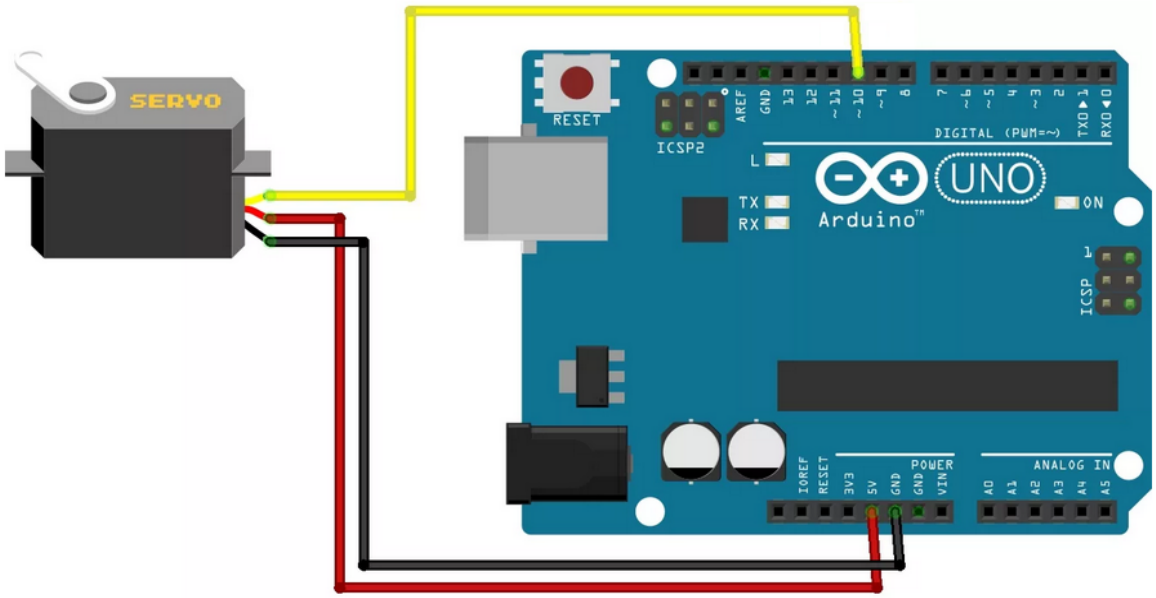
```
                            // und Sekunden RTC bzw. Temperatur und
                            // Luftfeuchtigkeit DHT11)
RTC.getTime(currentTime);
if (index == 1) Serial.println(currentTime.getHour());    // Stunden RTC
if (index == 2) Serial.println(currentTime.getMinutes()); // Minuten RTC
if (index == 3) Serial.println(currentTime.getSeconds()); // Sekunden RTC
if (index == 4) // Temperatur & Luftfeuchtigkeit
{
  byte newTemperature;
  byte newHumidity;
  if (dht11.read(&newTemperature, &newHumidity, NULL) == SimpleDHTErrSuccess)
{
  temperature = newTemperature;
  humidity = newHumidity;
}
Serial.println(100*int(temperature) + int(humidity)); // Temperatur &
                                                      // Luftfeuchte DHT11
delay(500);
```

6.2.4.5 Beispieldatei

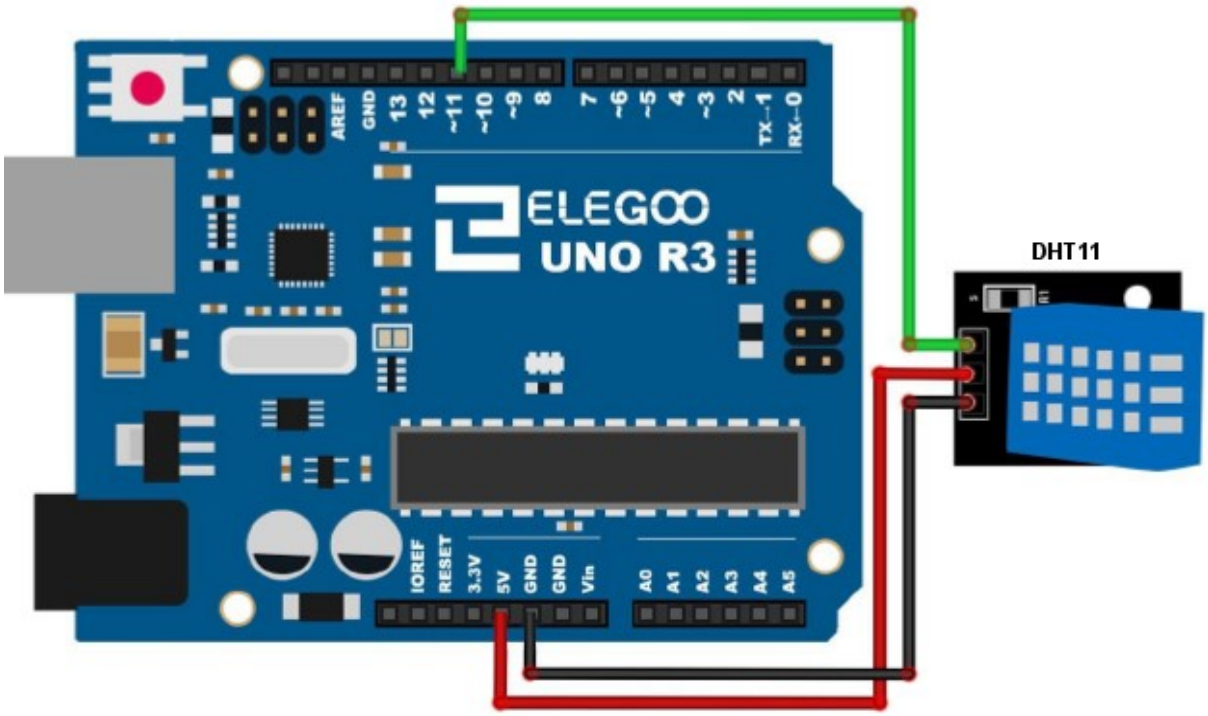
`ArduinoUnoR4-UserAnaIn.bsy`

7 Anhang

7.1 Anschlussbelegung Servo-Motor



7.2 Anschlussbelegung DHT11-Temperatursensor



7.3 Arduino Uno R4-Pinbelegung