

UnoArduSimV2.0.1 Vollständige Hilfe

Inhaltsverzeichnis

Überblick

CodeBereich, Präferenzen und Editieren / Ansehen

CodeBereich

Präferenz

Editieren / Ansehen

VariablenBereich und Editieren / Verfolgen Variable fenster

LaborBankBereich

Das 'Uno'

'I / O' Geräte

'Serial' Monitor ('SERIAL')

Software Serieller ('SFTSER')

SD-Laufwerk ('SD_DRV')

Schieberegister Sklave ('SRSLV')

Konfigurierbarer SPI-Sklave ('SPISLV')

Zweiadrige I2C-Sklave ('I2CSLV')

Schrittmotor ('STEPR')

Gleichstrom Motor ('MOTOR')

ServoMotor ('SERVO')

Digitaler Impulsgeber ('PULSER')

Generator Ein-Schuss ('1SHOT')

Analoger Funktionsgenerator ('FUNCGEN')

Piezoelektrischer Lautsprecher ('PIEZO')

Druckknopf ('PUSH')

Schiebeschalterwiderstand ('R = 1K')

Farbige LED ('LED')

Analoger Schieberegler

Menüs

Datei:

Lade INO oder PDE Programm (Strg-L)

Editieren / Ansehen (Strg-E)

Speichern

Speichern Als

Nächste ('#include')

Vorherige

Ausgang

Finden:

Nächste Funktion/Var

Vorherige Funktion/Var

Setze Suchtext (Strg-F)

Finde den Nächsten Text

Vorherigen Text Finden

Ausführen:

Schritt Hinein (F4)

Schritt Über (F5)

Schritt Aus (F6)

Ausführen Dort (F7)

Ausführen Bis(F8)

Ausführen (F9)

Halt (F10)

Zurücksetzen

Animieren

Zeitlupe

Optionen:

Schritt Über Strukturen / Operatoren

Register-Zuweisungsmodellierung

Fehler bei Nicht initialisiert

Hinzugefügt 'loop()' Verzögerung

Konfigurieren:

'I / O' Geräte

Präferenz

VarAktualisieren:

Erlaube Automatisches(-) Kontraktion

Erlaube Reduktion

Minimal

Änderungen Hervorheben

Fenster:

'Serial' Monitor

Alles Wiederherstellen

Pin DigitalWellenformen

Pin AnalogWellenform

Hilfe:

Schnellhilfe-Datei

Vollständige Hilfedatei

Fehlerbehebung

Änderung / Verbesserungen

Über

Modellieren

'Uno' Leiterplatte und 'I / O' Geräte

Zeitliche Koordinierung

'I / O' Gerätezeitsteuerung

Geräusche

Einschränkungen und nicht unterstützte Elemente

Enthaltene Dateien

Dynamische Speicherzuordnungen und RAM

'Flash' Speicherzuordnungen

'String' Variablen

Arduino-Bibliotheken

Zeiger

Objekte 'class' und 'struct'

Umfang

Qualifikanten 'unsigned', 'const', 'volatile', 'static'

Compiler-Richtlinien

Arduino-Sprachelemente

C / C ++ - Sprachelemente

Funktionsvorlagen

Echtzeit-Emulation

Versionshinweise

Fehlerbehebung

V2.0.1- Jan. 2018

V2.0- Dez. 2017

V1.7.2- Feb.2017

V1.7.1- Feb.2017

V1.7.0- Dez.2016

V1.6.3- September 2016

V1.6.2- September 2016

V1.6.1- Aug. 2016

V1.6 - Juni 2016

V1.5.1 - Juni 2016

[V1.5 - Mai 2016](#)
[V1.4.3 - Apr. 2016](#)
[V1.4.2 - März 2016](#)
[V1.4.1 - Jan. 2016](#)
[V1.4 - Dez. 2015](#)
[V1.3 - Okt. 2015](#)
[V1.2 - Juni 2015](#)
[V1.1 - März 2015](#)
[V1.0.2 - Aug. 2014](#)
[V1.0.1 - Juni 2014](#)

Änderungen / Verbesserungen

[V2.0.1- Jan. 2018](#)
[V2.0 Dez. 2017](#)
[V1.7.2- Feb. 2017](#)
[V1.7.1- Feb. 2017](#)
[V1.7.0- Dez.2016](#)
[V1.6.3- September 2016](#)
[V1.6.2- September 2016](#)
[V1.6.1- August 2016](#)
[V1.6 - Juni 2016](#)
[V1.5.1 - Juni 2016](#)
[V1.5 - Mai 2016](#)
[V1.4.2 - März 2016](#)
[V1.4 - Dez. 2015](#)
[V1.3 - Okt 2015](#)
[Version 1.2 Juni 2015](#)
[V1.1 - März 2015](#)
[V1.0.1 - Juni 2014](#)

Überblick

UnoArduSim ist ein Freeware- **Echtzeit**- Simulator-Tool (siehe Modellierung für Zeitliche Koordinierung- **Einschränkungen**), das ich für den Schüler und Arduino-Enthusiasten entwickelt habe. Es wurde entwickelt, um Ihnen zu ermöglichen, mit Arduino-Programmen zu experimentieren und diese leicht zu debuggen, **ohne dass Sie dafür die Hardware benötigen** . Es ist auf das **Arduino 'Uno'** Leiterplatte ausgerichtet und ermöglicht Ihnen die Auswahl aus einer Reihe von virtuellen 'I / O' -Geräten und die Konfigurierung und Verbindung dieser Geräte mit Ihrem virtuellen 'Uno' im **LaborBankBereich** . - Sie müssen sich keine Gedanken über Verdrahtungsfehler, unterbrochene / lose Verbindungen oder fehlerhafte Geräte machen, die Ihre Programmentwicklung und das Testen beeinträchtigen.

UnoArduSim bietet einfache Fehlermeldungen für alle Parsing- oder Ausführungsfehler, auf die es trifft, und ermöglicht das Debuggen mit **Zurücksetzen** , **Ausführen** , **Ausführen-Dort**, **Ausführen-Bis**- , **Halt**- und flexible **Schritt**- Vorgänge im **Codebereich** mit gleichzeitiger Anzeige aller globalen und derzeit aktiven lokalen Variablen, Arrays und Objekte im **Variablenbereich**. Laufzeit-Array-Grenzen Prüfung wird zur Verfügung gestellt, und ATmega RAM-Überlauf wird erkannt werden (und die Täter Programmzeile hervorgehoben!). Alle elektrischen Konflikte mit angeschlossenen 'I / O' -Geräten werden markiert und gemeldet, sobald sie auftreten.

Wenn ein INO oder PDE - Programmdatei geöffnet wird, wird es in das Programm - **CodeBereich** geladen. Das Programm wird dann analysiert, und „kompilierte“ in eine in Zeichen übersetzte ausführbaren welche für **simulierte Ausführung** dann bereit ist Jede parse Fehler (im Gegensatz zu Arduino.exe, eine eigenständige ausführbare Binärdatei wird *nicht* angelegt) wird erfasst und gekennzeichnet durch die Linie hervorgehoben , die fehlgeschlagen parsen und die Berichterstattung über die Fehler auf der **Statusleiste** am unteren Rand des UnoArduSim Fenster. Ein **Editieren / Ansehen**- Fenster kann geöffnet werden, damit Sie eine Syntax-hervorgehobene Version Ihres Benutzerprogramms sehen und bearbeiten können. Fehler während der simulierten Ausführung (z. B. nicht übereinstimmende Baudraten) werden in der Statusleiste und über eine Popup-Meldungsbox gemeldet.

UnoArduSim V2.0 ist eine im Wesentlichen vollständige Implementierung der **Arduino Programmiersprache V1.6.6 wie im. dokumentiert arduino.cc** . Sprachreferenz-Webseite und mit Hinzufügungen wie in der Versions-Herunterladen-Seite Release Notes angegeben. Obwohl UnoArduSim die vollständige C ++ - Implementierung des zugrunde liegenden GNU-Compilers Arduino.exe nicht unterstützt, ist es wahrscheinlich, dass nur die fortgeschrittensten Programmierer feststellen würden, dass einige C / C ++ - Elemente, die sie verwenden möchten, fehlen (und natürlich immer einfach sind) Arbeitsumgebungen für solche fehlenden Funktionen kodieren. Im Allgemeinen habe ich nur das unterstützt, was ich für die nützlichsten C / C ++ - Funktionen für Arduino-Bastler und Schüler **halte** - zum Beispiel werden **'enum'** und **'#define'** unterstützt, aber Funktionszeiger nicht. Obwohl benutzerdefinierte Objekte (**'class'** und **'struct'**) und (die meisten) Operatorüberladungen unterstützt werden, ist die **Mehrfachvererbung nicht möglich** .

Da UnoArduSim ein High-Level-Language-Simulator ist, werden **nur C / C ++ - Anweisungen unterstützt** , **Assemblersprachenanweisungen jedoch nicht** . Da es sich nicht um eine Low-Level-Maschinensimulation handelt, sind die **ATmega328-Register für Ihr Programm** weder zum Lesen noch zum Schreiben **zugänglich** , obwohl Registerzuweisung, Weitergabe und Rückgabe emuliert werden (wählen Sie dies unter dem Menü **Optionen**).

Ab V2.0 verfügt UnoArduSim über eine integrierte automatische Unterstützung für eine begrenzte Teilmenge der von Arduino bereitgestellten Bibliotheken: **'Stepper.h'** , **'SD.h'** , **'Servo.h'** , **'SoftwareSerial.h'** , **'SPI.h'** , **'Wire.h'** und **'EEPROM.h'** (version 2). Für beliebige **'#include'** von Benutzern erstellte Bibliotheken wird UnoArduSim **nicht** die übliche Arduino-Installationsverzeichnisstruktur durchsuchen, um die Bibliothek zu finden; stattdessen **müssen** Sie die entsprechenden Header („.h“) und Quelle („.CPP“) Datei in das gleiche Verzeichnis wie die Programmdatei kopieren , die Ihr auf (natürlich vorbehaltlich der Einschränkung arbeiten , dass der Inhalt jeder **'#include'** datei muss für den UnoArduSim-Parser vollständig verständlich sein.

Ich habe UnoArduSimV2.0 in QtCreator mit mehrsprachiger Unterstützung entwickelt und ist derzeit nur für Windows™ verfügbar. Die Portierung auf Linux oder MacOS ist ein Projekt für die Zukunft. UnoArduSim ist aus Simulatoren hervorgegangen, die ich im Laufe der Jahre für Kurse, die ich an der Queen's University unterrichtet habe, entwickelt hatte, und es wurde einigermaßen ausführlich getestet, aber es wird sicherlich noch ein paar Bugs geben, die sich dort verstecken. Wenn Sie einen Fehler melden möchten, beschreiben Sie ihn bitte (kurz) in einer E-Mail an unoArduSim@gmail.com und stellen Sie **sicher, dass Sie den vollständigen Arduino-Quellcode für den Programmfehler anhängen**, damit ich den Fehler replizieren und beheben kann. Ich werde nicht auf einzelne Fehlerberichte antworten, und ich habe keine garantierten Zeitpläne für Korrekturen in einer nachfolgenden Version (denken Sie daran, es gibt fast immer Workarounds!).

Prost,

Stan Simmons, Ph. D, P. Eng.
Assoziierter Professor (im Ruhestand)
Fakultät für Elektrotechnik und Informationstechnik
Königin-Universität
Kingston, Ontario, Kanada

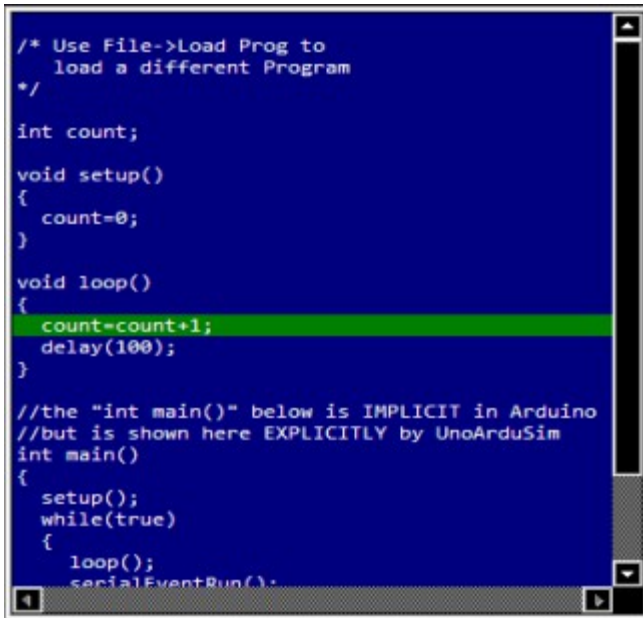


CodeBereich, Präferenzen und Editieren / Ansehen

(Nebenbei: Die unten gezeigten Beispielfenster sind alle unter einem vom Benutzer gewählten Windows-OS-Farbschema mit einer dunkelblauen Fensterhintergrundfarbe).

CodeBereich

Der **CodeBereich** zeigt Ihr Benutzerprogramm an, und die Hervorhebung verfolgt dessen Ausführung. Nachdem ein geladenes Programm erfolgreich analysiert wurde, wird die erste Zeile angezeigt 'main()' ist markiert und das Programm ist zur Ausführung bereit. Beachten Sie, dass 'main()' wird implizit von Arduino (und von UnoArduSim) hinzugefügt und Sie enthalten es **nicht** als Teil Ihrer Benutzerprogrammdatei. Die Ausführung ist unter der Kontrolle des Menüs **Ausführen**, und die dazugehörigen **Tool-Bar** Tasten und Funktionstastenkürzel.







```
/* Use File->Load Prog to
   load a different Program
*/

int count;

void setup()
{
  count=0;
}

void loop()
{
  count=count+1;
  delay(100);
}

//the "int main()" below is IMPLICIT in Arduino
//but is shown here EXPLICITLY by UnoArduSim
int main()
{
  setup();
  while(true)
  {
    loop();
    serialEventRun();
  }
}
```



Nach der schrittweisen Ausführung durch eine (oder mehrere) Anweisungen (Sie können **Tool-Bar**-Schaltflächen verwenden , , , oder ), wird die Programmzeile, die als nächstes ausgeführt wird, hervorgehoben - die markierte Zeile ist immer die nächste Zeile, **die zur Ausführung bereit ist**.





Wenn ein laufendes Programm einen (temporären **Ausführen-Dort**) Haltepunkt erreicht, wird die Ausführung angehalten und die Haltepunktlinie wird markiert (und ist dann für die Ausführung bereit).

Wenn die Programmausführung derzeit angehalten ist und Sie im **Fenster "CodeBereich"** klicken, **wird die gerade angeklickte Zeile hervorgehoben**. Dies ändert jedoch *nicht* die aktuelle Programmzeile, was die Programmausführung betrifft. Sie können jedoch bewirken, dass die Ausführung *bis zu* der gerade markierten Zeile *fortschreitet*, indem Sie auf

Ausführen klicken  **Werkzeugleisten**-Taste. Mit

dieser Funktion können Sie schnell und einfach bestimmte Zeilen in einem Programm erreichen, so dass Sie anschließend Zeile für Zeile über einen Programmabschnitt von Interesse springen können.

Wenn Ihr geladenes Programm irgendwelche '#include' -Dateien hat, können Sie mit zwischen ihnen wechseln von Datei | Vorherige und Datei | Nächste (mit Werkzeugleisten-Schaltflächen  und ).

Das Menü **Suche** ermöglicht es Ihnen, schnell im **CodeBereich** zwischen **Funktionen** zu **springen** (nach dem ersten einer Zeile klicken darin, es zu konzentrieren), indem Sie die **Next** und **Previous** - Befehle (mit **Tool-Bar** Tasten  und  oder Tastaturkürzel **PgDn** und **PgUp**). Alternativ können Sie einen bestimmten Text mit seinen Textbefehlen (mit den Symbolleisten-Schaltflächen  und ) finden und oder Tastenkombinationen nach oben und unten), nachdem Sie zuerst **Finden** | **Setze Suchtext** oder Symbolleiste-Schaltfläche fest.

Präferenz



Konfigurieren | Präferenzen ermöglicht Benutzern um Programm- und Anzeigeeinstellungen festzulegen (die ein Benutzer normalerweise bei der nächsten Sitzung annehmen möchte). Diese können daher gespeichert und aus einer **myArduPrefs.txt**- Datei geladen werden, die sich im selben Verzeichnis wie das geladen 'Uno' -Programm befindet (**myArduPrefs.txt** wird automatisch geladen, wenn es existiert).

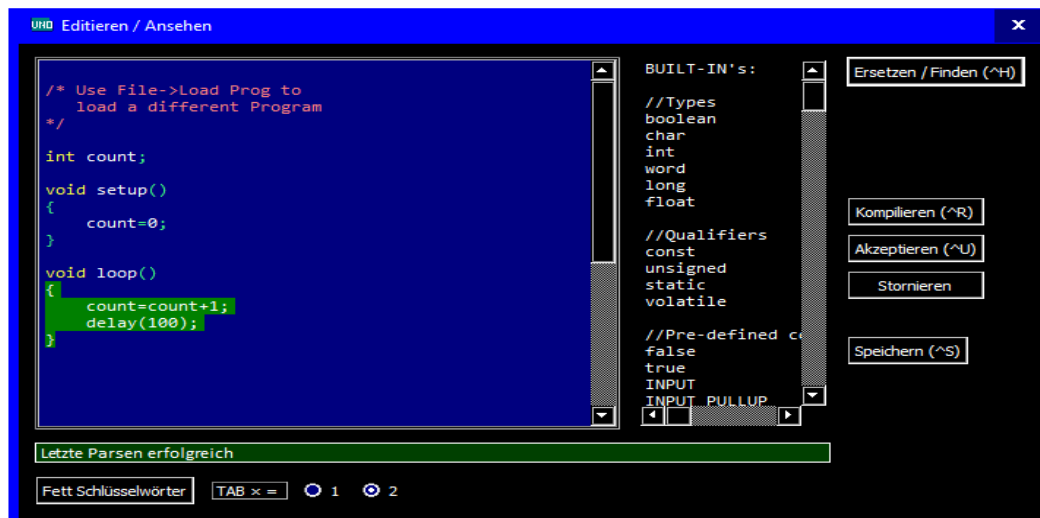
In diesem Dialogfeld können Sie zwischen zwei mono-independent-Schriftarten und drei Schriftgradtypen sowie weiteren Voreinstellungen wählen. Ab V2.0 ist nun auch der Sprach-Chpice enthalten. - Dies beinhaltet immer Englisch (

en), plus ein oder zwei andere Sprachen für die Benutzersprache (wo diese existieren), und eine Überschreibung basierend auf dem **zweistelligen** ISO-639 Sprachcode *in der ersten Zeile* der **myArduPrefs.txt** Datei (if man ist dort vorgesehen). Die Auswahlmöglichkeiten erscheinen nur, wenn eine Übersetzungsdatei ".qm" im Übersetzungsordner (im UnoArduSim.exe Home-Verzeichnis) vorhanden ist.

Editieren / Ansehen

Durch einen Doppelklick auf eine beliebige Zeile im **CodeBereich** (oder über das **Menü Datei**) wird ein **Editieren/ Ansehen**- Fenster geöffnet, in dem Änderungen an Ihrer Programmdatei vorgenommen werden können, wobei die aktuell ausgewählte Zeile im CodeBereichmarkiert ist.

Dieses Fenster verfügt über eine vollständige Bearbeitungsfunktion mit dynamischer Syntax-Hervorhebung



(verschiedene Hervorhebungsfarben werden für C ++ - Schlüsselwörter, Kommentare usw. verwendet). Es gibt eine optionale Fettschrift-Hervorhebung und automatische Formatierung der Einzugsebene (vorausgesetzt, Sie haben dies über Konfigurieren | Präferenzen ausgewählt). Sie können auch bequem integrierte Funktionsaufrufe (oder integrierte '#define' -Konstanten) auswählen, die Sie aus der bereitgestellten Liste in Ihr Programm einfügen möchten - doppelklicken Sie einfach auf das gewünschte Listenfeld, um es hinzuzufügen Ihr Programm an der aktuellen Cursorposition (Funktionsaufruf variablen - **Typen** sind nur zur Information und werden , wenn zu Ihrem Programm hinzugefügt Dummy - Platzhalter leaveie gezupft).

Das Fenster hat die Funktionen **Finden** (**Strg-F** verwenden) und **Suchen / Ersetzen** (**Strg-H** verwenden) . Das **Editieren / Ansehen**- fenster hat die **Schaltflächen Zurücknehmen** (**Strg-Z**) und **Wiederholen** (**Strg-Y**) (die automatisch erscheinen).

Um **alle Änderungen** zu verwerfen, **die** Sie seit dem ersten Öffnen des Programms zur Bearbeitung vorgenommen haben, klicken Sie auf die Schaltfläche **Abbrechen**. Um den aktuellen Status zu akzeptieren, klicken Sie auf die Schaltfläche **Akzeptieren** und das Programm wird automatisch erneut analysiert (und heruntergeladen zu dem 'Uno' wenn keine Fehler gefunden werden) und der neue Status wird im Haupt UnoArduSim Fenster **Status-Bar**.

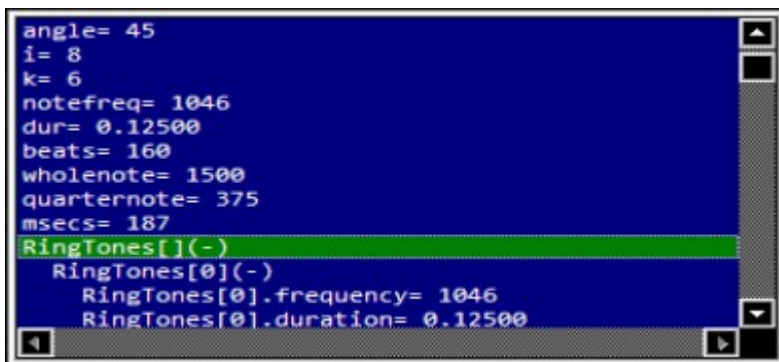
Eine **Compile (Strg-R)** -Schaltfläche (plus eine zugehörige **Parse-Status-** Message-Box, wie im Bild oben zu sehen) wurde hinzugefügt, um das Testen von Änderungen zu ermöglichen, ohne das Fenster zu schließen. Eine Schaltfläche **Speichern (Strg-S)** wurde auch als Verknüpfung hinzugefügt (entspricht einem **Akzeptieren** und einem späteren separaten **Speichern** aus dem Hauptfenster).

Bei **Stornieren** oder **Accept** ohne vorgenommene Änderungen ändert sich die aktuelle Zeile des **CodeBereich** in die **letzte Caret-Position "Ansehen / Editieren"**, und Sie können diese Funktion verwenden, um den **CodeBereich** auf eine bestimmte Zeile zu springen (möglicherweise um sich auf die **Ausführung vorzubereiten**). **Zu**), Sie können auch **Strg-PgDn** und **Strg-PgUp** verwenden, um zur nächsten (oder vorherigen) Leerzeilenunterbrechung in Ihrem Programm zu springen - dies ist nützlich für schnelles Navigieren zu wichtigen Stellen (wie Leerzeilen zwischen Funktionen). Sie können auch **Strg-Home** und **Strg-End** benutzen, um zum Programmstart zu springen bzw. zu beenden.

Die **automatische Formatierung der Formatierung auf Tab-Ebene** erfolgt beim **Öffnen** des Fensters, wenn diese Option unter **Konfigurieren | Präferenzen**. Sie können Tabs auch selbst zu einer Gruppe vorher ausgewählter aufeinanderfolgender Zeilen hinzufügen oder löschen, indem Sie den **rechten** oder den **linken Pfeil** der Tastatur verwenden. Die **automatische Einrückung muss** jedoch **deaktiviert sein, um** zu vermeiden, dass Sie Ihre eigenen Tab-Level verlieren.






Damit Sie Ihre Kontexte und geschweiften GeschweifteKlammern besser verfolgen können, wird durch Klicken auf eine '{' oder '}' GeschweifteKlammer der **gesamte Text zwischen dieser GeschweifteKlammer und dem entsprechenden Partner hervorgehoben**.

VariablenBereich und Editieren / Verfolgen Variable fenster



Der **VariablenBereich** befindet sich direkt unter dem **CodeBereich**. Es zeigt die aktuellen Werte für jede Benutzer globale und aktive (in-scope) lokale Variable / Array / Objekt im geladenen Programm. Wenn sich Ihre Programmausführung zwischen Funktionen bewegt, **ändern sich die Inhalte so, dass sie nur die lokalen Variablen widerspiegeln, auf die die aktuelle Funktion / der aktuelle Bereich zugreifen kann, sowie alle vom Benutzer deklarierten Globals**. Alle Variablen, die

als **'const'** oder als **'PROGMEM'** (zugewiesen für 'Flash' datenbank) deklariert sind, haben Werte, die sich nicht ändern können, und um Platz zu sparen, werden diese daher **nicht angezeigt**. **'Servo'** und **'SoftwareSerial'** Objektinstanzen enthalten keine nützlichen Werte und werden daher auch nicht angezeigt.

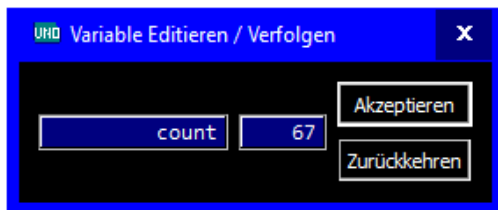
Die Befehle im Menü **Finden** (mit den Symbolleisten- Schaltflächen  und  Oder Tastenkombinationen **PgDn** und **PgUp**) können **springen** schnell in der **VariablenBereich** zwischen **variablen** (nach dem Sie ersten Klick darauf, um es zu aktivieren). Alternativ können Sie einen bestimmten Text mit seinen Textsuchbefehlen (mit den Symbolleisten- Schaltflächen  und , oder Tastenkombinationen nach oben und unten), nach dem ersten Gebrauch **Finden | Setze Suchtext**, oder .

Arrays und **Objekte** werden entweder im **nicht expandierten** oder im **erweiterten** Format mit einem nachgestellten Pluszeichen (+) bzw. Minuszeichen (-) angezeigt. Das Symbol für ein Array **x** zeigt als **x[]**. Wenn Sie alle Elemente des Arrays anzeigen möchten, klicken Sie einfach im VariablenBereich auf **x[] (+)**. Um zu einer nicht expandierten Ansicht zurückzukehren, klicken Sie auf das **x[] (-)**. Der nicht erweiterte Standard für ein Objekt **p1** zeigt als **p1 (+)**. Um es so zu erweitern, dass alle Mitglieder dieser 'class' oder 'struct' -Instanz angezeigt werden, klicken Sie einmal auf **p1 (+)** im **VariablenBereich**. Um zu einer nicht expandierten Ansicht zurückzukehren, klicken Sie einfach auf **p1 (-)**.

Wenn Sie **auf eine Zeile klicken, um sie hervorzuheben** (dies kann eine einfache Variable oder die Aggregat (+) oder (-) Zeile eines Arrays oder Objekts oder eines einzelnen Array-Elements oder Objektelements sein), führen Sie einen **Ausführen-Bis - Die Ausführung wird fortgesetzt** und bei dem nächsten **Schreibzugriff** innerhalb des ausgewählten Aggregats oder an dem ausgewählten einzelnen Variablenspeicherort angehalten.

Bei Verwendung von **Schritt** oder **Ausführen** werden Aktualisierungen der angezeigten Variablenwerte gemäß den Benutzereinstellungen im Menü **VarAktualisieren** vorgenommen. Dies ermöglicht ein vollständiges Verhalten von minimalen periodischen Aktualisierungen bis zu vollständigen sofortigen Aktualisierungen. Reduzierte oder minimale Aktualisierungen sind nützlich, um die CPU-Last zu reduzieren, und können erforderlich sein, um zu verhindern, dass die Ausführung in Echtzeit unter einem ansonsten übermäßigen **Aktualisierungsaufwand** für **VariablenBereich**. Wenn **Animate aktiv** ist oder die **Änderungen Hervorheben** ausgewählt ist, wird der Wert einer Variablen während der **Ausführung** sofort aktualisiert, und der angezeigte Wert wird hervorgehoben. Dies führt dazu, dass der **VariablenBereich** scrollt (wenn benötigt) zu der Zeile, die diese Variable enthält, und die Ausführung wird nicht mehr in Echtzeit erfolgen!

Wenn die **Ausführung** nach dem **Schritt** gefriert, **Ausführen-Dort**, **Ausführen-Bis**, oder **Ausführen -dann-Halt**, hebt die **VariablenBereich** die Variable an die **Adresse Ort** entspricht (**en**), die während dieser Ausführung (falls vorhanden) durch den **allerletzten Befehl modifiziert wurde** (einschließlich durch Initialisierung von Variablendeklarationen). Wenn diese Anweisung ein **Objekt oder Array vollständig** gefüllt hat, wird die **übergeordnete (+) oder (-) Zeile** für dieses Aggregat hervorgehoben. Wenn stattdessen der Befehl einen momentan sichtbaren Standort ändert, wird er hervorgehoben. Aber wenn die modifizierte Stelle (n) ist (sind) versteckt derzeit innerhalb eines nicht erweiterten Array oder ein Objekt, das Aggregat **Elternlinie** eine **kursive Schrift highlighting** als visueller Hinweis bekommt, dass etwas in ihm geschrieben wurde - zu klicken, um es zu erweitern wird dann bewirkt, dass das **zuletzt** geänderte Element oder Element markiert wird.



Das **Variable Editieren / Verfolgen** Fenster gibt Ihnen die **Möglichkeit, einen beliebigen Variablenwert während der Ausführung zu verfolgen** oder **seinen Wert in der Mitte der (angehaltenen) Programmausführung zu ändern** (damit Sie testen können, welchen Effekt es hat, mit diesem neuen Wert fortzufahren). **Halten Sie die** Ausführung zunächst an und **klicken Sie dann mit der linken Maustaste doppelt** auf die

Variable, deren Wert Sie verfolgen oder ändern möchten. Um den Wert während der Programmausführung einfach zu überwachen, **lassen Sie den Dialog offen** und dann einen der Befehle **Ausführen** oder **Schritt** - sein Wert wird in **Editieren / Verfolgen** gemäß den gleichen Regeln aktualisiert, die für Updates im **VariablenBereich** gelten. **Um den Variablenwert zu ändern**, füllen Sie **den Wert** der Eingabefelder und **Akzeptieren**. Weiter Ausführung (eine des **Schritt** oder **Ausführen** - Befehle) nach vorne, dass neuen Wert von diesem Punkt zu verwenden (oder Sie können auf den vorherigen Wert **Revert**).

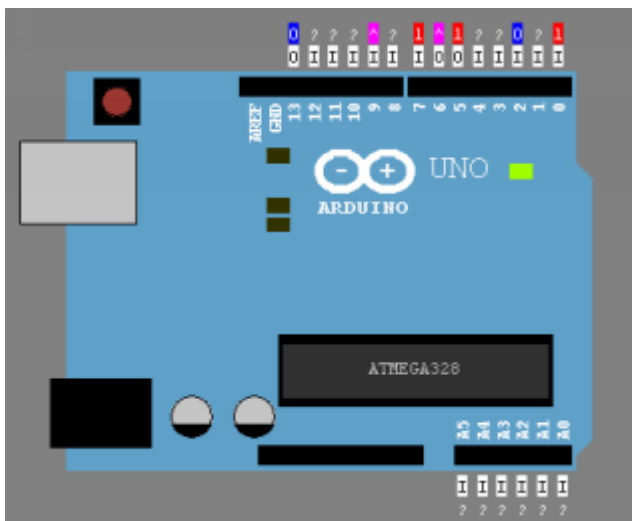
Beim Laden oder Zurücksetzen des Programms beachten Sie, dass alle **nicht initialisierten Wertvariablen** auf den Wert 0 zurückgesetzt werden und alle **nicht initialisierten Zeigervariablen** auf 0x0000 zurückgesetzt werden.

LaborBankBereich

Der LaborBankBereich zeigt eine 5-Volt 'Uno' -Leiterplatte, die von einer Reihe von 'I / O' -Geräten umgeben ist, die Sie auswählen / anpassen können, und stellen Sie eine Verbindung zu Ihren gewünschten 'Uno' -Pins her.

Das 'Uno'

Dies ist eine Darstellung des 'Uno' Leiterplattes und seiner integrierten LEDs. Wenn Sie ein neues Programm in UnoArduSim laden, wird es, wenn es erfolgreich analysiert wurde, einem "simulierten heruntergeladen" unterzogen, der die Art und Weise simuliert, wie sich ein tatsächliches 'Uno' Leiterplatte verhält - Sie werden die serielle RX und TX LED blinken sehen (zusammen mit Aktivität auf den Pins 1 und 0, die *für die serielle Kommunikation mit einem Host-Computer fest verdrahtet* sind). Dies ist unmittelbar gefolgt von einem Pin 13 LED-Blitz, der das Zurücksetzen der Karte und (und UnoArduSim automatisches Anhalten an) den Beginn der geladenen Programmausführung anzeigt. Sie können diese Anzeige, und die damit verbundene Ladeverzögerung, vermeiden, indem deaktivieren Sie Zeigen Herunterladen von Konfigurieren | Präferenzen.



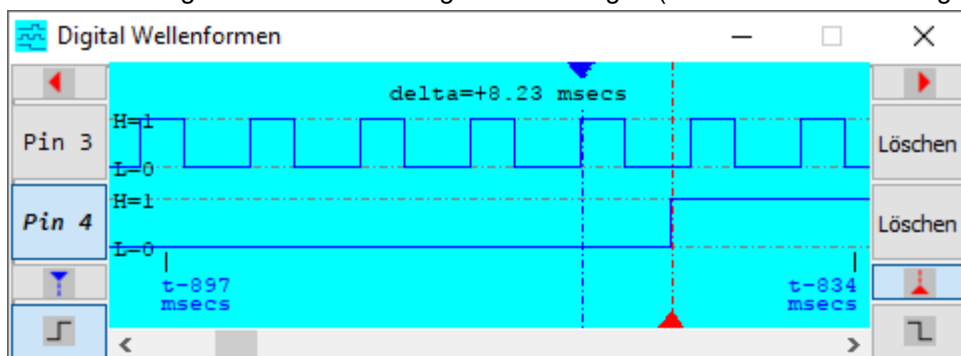
In diesem Fenster können Sie die digitalen Logikpegel an allen 20 'Uno' -Pins ('1' auf Rot für 'HIGH' , '0' auf Blau für 'LOW' und '?' auf grau für eine undefinierte unbestimmte Spannung) und programmierte Richtungen ('I' für 'INPUT' , oder 'O' für 'OUTPUT'). Für Pins, die mit PWM über 'analogWrite()' oder 'tone()' oder 'Servo.write()' **gepulst werden**, Die Farbe ändert sich in Violett und das angezeigte Symbol wird zu '^' .

Beachten Sie, dass die **digitalen Pins 0 und 1 über 1-kOhm-Widerstände für die serielle Kommunikation mit einem Host-Computer fest mit dem USB-Chip verbunden sind**.









Nebenbei: Die digitalen Pins 0-13 erscheinen als Simulatorpins 0-13 und die analogen Pins 0-5



erscheinen als A0-A5. Um auf einen analogen Pin in Ihrem Programm zuzugreifen, können Sie sich auf die Pin-Nummer durch einen von zwei äquivalenten Zahlensätzen beziehen: 14-19; oder A0-A5 (A0-A5 sind in 'const' Variablen mit den Werten 14-19 eingebaut). Und nur wenn 'analogRead()' verwendet wird, wird eine dritte Option verfügbar gemacht - Sie können für diese eine Anweisung das Präfix 'A' von der Pin-Nummer löschen und einfach 0-5 verwenden. Um auf die Pins 14-19 in Ihrem Programm mit 'digitalRead()' oder 'digitalWrite()' zuzugreifen, können Sie einfach auf diese Pin-Nummer verweisen, oder Sie können stattdessen die Aliasnamen A0-A5 verwenden.

Ein Linksklick auf einen beliebigen 'Uno' -Pin öffnet ein **Pin-DigitalWellenformen-** Fenster, in dem die **Aktivität** des **digitalen Levels in** der letzten **Sekunde für diesen Pin** angezeigt wird . Sie können auf andere Pins klicken, um diese zum Pin DigitalWellenformen-Zeigen hinzuzufügen (bis zu 4 Wellenformen gleichzeitig).



Klicken Sie links oder rechts auf die Seitenansicht, oder verwenden Sie die Tasten Home, Bild auf, Bild ab, Ende

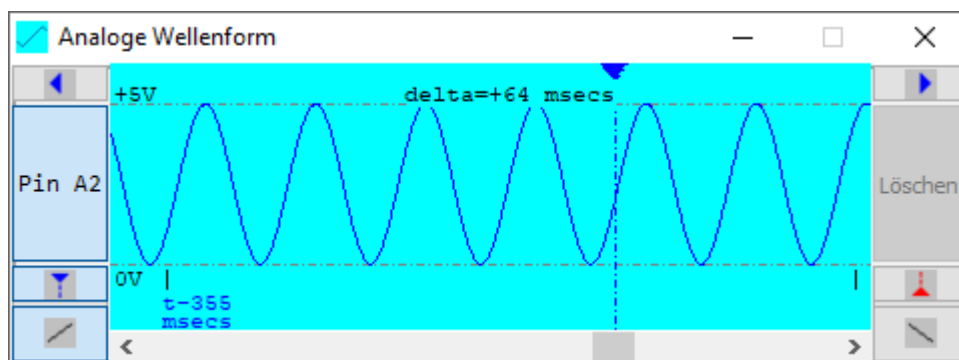
Eine der angezeigten Wellenformen ist die aktive Pin- Wellenform , angezeigt durch ihre "**Pin**" -Taste, die als gedrückt angezeigt wird (wie in der obigen Pin-DigitalWellenformen Bildschirmaufnahme). Sie können eine Wellenform auswählen, indem Sie auf die Schaltfläche "**Pin-Nummer**" klicken und dann die interessierende Kantenpolarität auswählen, indem Sie auf den entsprechenden Auswahlknopf für die ansteigende / fallende Kantenpolarität klicken. , oder  oder mithilfe der Tastenkombination **Pfeil nach oben** und **Pfeil nach unten** . Sie können dann den aktiven Cursor (entweder die blaue oder die rote Cursor-Linie mit ihrer Delta-Zeit) rückwärts oder vorwärts zur ausgewählten digitalen Kante *dieses aktiven Pins springen* Wellenform mit den Cursortasten (,  oder ,  , abhängig davon, welcher Cursor zuvor mit aktiviert wurde  oder ), oder benutzen Sie einfach die Tastaturtasten \leftarrow und \rightarrow .

Um einen Cursor zu aktivieren, klicken Sie auf den farbigen Aktivierungsknopf ( oder  siehe oben) - *blättert auch die Ansicht an die aktuelle Position des Cursors* . Alternativ können Sie die Aktivierung zwischen Cursors (mit ihren jeweils zentrierten Ansichten) schnell mit der Tastenkombination **TAB** umschalten.







Sie können den aktuell aktivierten Cursor **springen**, indem Sie mit der **linken Maustaste auf eine beliebige Stelle** in der Wellenformansicht auf dem Bildschirm **klicken** . Alternativ können Sie entweder die rote oder die blaue Cursorlinie auswählen, indem Sie mit der rechten Maustaste darauf klicken (um sie zu aktivieren), und dann auf den Cursor *ziehen ein neuer Standort* und Freigabe. Wenn sich ein gewünschter Cursor derzeit außerhalb des Bildschirms befindet, können Sie mit der **rechten** Maustaste auf eine beliebige **Stelle** in der Ansicht **klicken** , um zu dem neuen Bildschirmstandort zu springen. Wenn beide Cursor bereits auf dem Bildschirm angezeigt werden, wechselt der Rechtsklick einfach zwischen den aktivierten Cursor.

Um HINEINZOOMEN und RAUSZOOMEN (Zoom ist immer auf den ACTIVE-Cursor zentriert) zu drehen, verwenden Sie das Mausrad oder die Tastenkombination STRG-NACH-OBEN und STRG-AB-Pfeil .

Tun Sie stattdessen einen **Rechtsklick Auf jedem 'Uno' -Pin** öffnet sich ein **Pin-AnalogWellenform** Fenster, in dem die **Aktivitätsstufe** der **analogen Aktivität** auf der **letzten Sekunde** für diesen Pin angezeigt wird . Im Gegensatz zum Fenster "Pin DigitalWellenformen" können Sie analoge Aktivitäten jeweils nur auf einem Pin anzeigen.

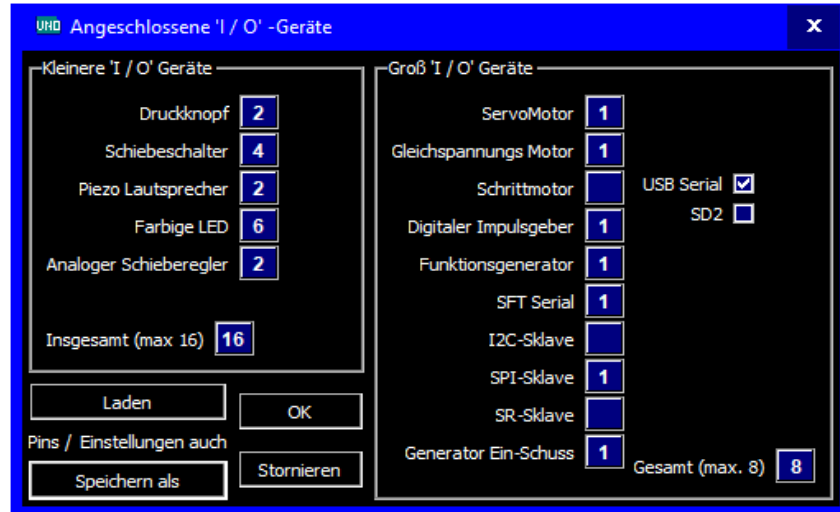


Klicken Sie links oder rechts auf die Seitenansicht, oder verwenden Sie die Tasten Home, Bild auf, Bild ab, Ende

Sie können *die* blauen oder roten Cursorlinien mit den Vorwärts- oder Rückwärtspfeiltasten zum nächsten ansteigenden oder fallenden "Steigungspunkt" **springen** (,  oder ,  , wieder abhängig vom aktivierten Cursor, oder benutzen Sie die \leftarrow und \rightarrow Tasten) in Verbindung mit den ansteigenden / absteigenden Auswahlstasten ,  (Der "Steigungspunkt" tritt auf, wenn die analoge Spannung durch den hohen digitalen Logikpegel-Schwellenwert des ATmega-Pins geht). Alternativ können Sie erneut klicken, um zu springen, oder diese Cursorlinien ähnlich wie im Fenster Pin DigitalWellenformen ziehen

'I / O' Geräte

Eine Reihe verschiedener Geräte umgibt das 'Uno' am Umfang des **LaborBankBereich** . "Kleine" 'I / O' -Geräte (von denen insgesamt maximal 16 möglich sind) befinden sich entlang der linken und rechten Seite des Fensters. "Große" 'I / O' -Geräte (von denen insgesamt bis zu 8 erlaubt sind) haben "aktive" Elemente und befinden sich am oberen und unteren Rand des **LaborBankBereich** . Die gewünschte Anzahl von jedem verfügbaren 'I / O' Gerät kann über das Menü **Configure | 'I / O' Geräte** .



Jedes 'I / O' -Gerät verfügt über einen oder mehrere Pin-Anschlüsse, die als **zweistellige** Pin-Nummer (00, 01, 02, ... 10,11,12, 13 und entweder A0-A5 oder 14-19) angezeigt werden. in einer entsprechenden Editierbox .. Für die Pin-Nummern 2 bis 9 können Sie einfach die einzelne Ziffer eingeben - die führende 0 wird automatisch bereitgestellt, aber für die Pins 0 und 1 müssen Sie zuerst die führende 0 eingeben. Die Eingänge befinden sich *normalerweise* auf der linken Seite eines 'I / O' -Gerätes und die Ausgänge befinden sich *normalerweise* auf der rechten Seite (*Platz ist zulässig*). Alle 'I / O' -Geräte reagieren direkt auf Pin-Level und Pin-Level-Änderungen und reagieren entweder auf Bibliotheksfunktionen, die auf ihre angeschlossenen Pins gerichtet sind, oder auf programmierte '**digitalWrite()**' (für "Bit-Banged" -Operation) .

Sie können mehrere Geräte an denselben ATmega-Pin anschließen, solange dies keinen **elektrischen Konflikt verursacht** . Ein solcher Konflikt kann entweder durch einen 'Uno' -Pin als '**OUTPUT**' **verursacht werden** , der gegen ein stark leitendes (niederohmiges) verbundenes Gerät fährt (z. B. gegen einen 'FUNCGEN' -Ausgang oder einen 'MOTOR' -Codiererausgang). oder durch zwei miteinander verbundene Geräte, die miteinander konkurrieren (z. B. sowohl eine 'PULSER' - als auch eine 'PUSH' - Taste, die an denselben Pin angeschlossen sind). Solche Konflikte würden bei einer echten Hardware-Implementierung katastrophal sein und sind daher nicht erlaubt und werden dem Benutzer über eine Popup-Meldungsbox angezeigt.

Der Dialog kann verwendet werden, damit der Benutzer die Typen und Nummern der gewünschten 'I / O' -Geräte auswählen kann. In diesem Dialogfeld können Sie auch 'I / O' Geräte in eine Textdatei speichern und / oder 'I / O' Geräte aus einer zuvor gespeicherten (oder bearbeiteten) Textdatei laden (einschließlich aller PIN-Anschlüsse und anklickbaren Einstellungen und Eingaben Werte).

Beachten Sie, dass ab Version 1.6 die Werte in den Editierfeldern Periode, Verzögerung und Impulsbreite in den jeweiligen IO-Geräte mit dem Buchstaben 'S' (oder 's') versehen werden können. Das zeigt an, dass sie entsprechend der Position eines globalen '**I skaliert**' werden sollen / **O ____ S'** Schieberegler , die **Tool-Bar** im Hauptfenster angezeigt wird . Wenn der Schieberegler ganz nach rechts steht, ist der Skalierungsfaktor 1,0 (Einheit), und bei vollständig nach links verschobenem Schieberegler beträgt der Skalierungsfaktor 0,0 (abhängig von den Mindestwerten, die von jedem einzelnen 'I / O' -Gerät erzwungen werden). Sie können mit diesem Schieberegler mehrere Werte **gleichzeitig** bearbeiten . Mit dieser Funktion können Sie den Schieberegler während der Ausführung ziehen, um die sich ändernden Impulsbreiten, -perioden und -verzögerungen für die angeschlossenen 'I / O' -Geräte einfach zu emulieren.

Der Rest dieses Abschnitts enthält Beschreibungen für jeden Gerätetyp.

'Serial' Monitor ('SERIAL')

Dieses 'I / O' -Gerät ermöglicht ATmega hardwaremäßige serielle Eingabe und Ausgabe (über den 'Uno' USB-Chip) an 'Uno' -Pins 0 und 1. Die Baudrate wird über die Dropdown-Liste am unteren Rand eingestellt. Die ausgewählte Baudrate **muss mit** dem Wert **übereinstimmen**, den Ihr Programm an die `'Serial.begin()'` -Funktion für die ordnungsgemäße Übertragung / Empfang **übergibt**. *Die serielle Kommunikation ist auf 8 Datenbits, 1 Stoppbit und kein Paritätsbit festgelegt.*

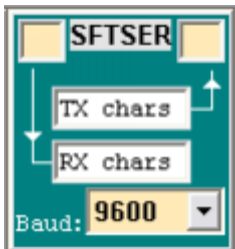


Um die Tastatureingabe zu Ihrem Programm, geben Sie ein oder mehrere Zeichen in der oberen (TX Zeichen) bearbeiten Fenster zu senden und dann drücken Sie die Eingabetaste. (Zeichen werden kursiv formatiert, um darauf hinzuweisen, dass Übertragungen begonnen haben) - oder wenn bereits Zeichen eingegeben werden, werden kursive Zeichen kursiv dargestellt. Sie können dann die `'Serial.available()'` - und `'Serial.read()'` -Funktionen verwenden, um die Zeichen in der Reihenfolge zu lesen, in der sie in den Pin-0-Puffer **empfangen wurden** (das Zeichen vom **Typ ganz** links wird zuerst gesendet). Formatierte Text- und numerische Ausdrücke oder unformatierte Bytewerte können durch Aufruf der Funktionen Arduino `'print()'`, `'println()'` oder `'write()'` zum unteren Konsolenausgabefenster (RX-Zeichen) **gesendet** werden.

Zusätzlich kann **ein größeres Fenster zum Einstellen / Anzeigen von TX- und RX-Zeichen durch Doppelklick (oder Rechtsklick) auf dieses 'Serial' -Gerät geöffnet werden**. Dieses neue Fenster hat eine größere TX-Zeichen-Editierbox und eine separate 'Send' -Schaltfläche, die angeklickt werden kann, um die TX-Zeichen an die 'Uno' (an Pin 0) zu senden. Es gibt auch eine Option zum erneuten Interpretieren \-vorfixiert Zeichen wie `'\n'` oder `'\t'` für nicht-rohe Anzeige

Software Serieller ('SFTSER')

Dieses 'I / O' -Gerät ermöglicht bibliothekssoftwarevermittelte oder alternativ "bit-banged", serielle Eingabe und Ausgabe auf jedem Paar von 'Uno' -Pins, die Sie ausfüllen möchten (mit **Ausnahme der** Pins 0 und 1, die sind für hardware `'Serial'` communication **reserviert**. Ihr Programm muss eine `'#include <SoftwareSerial.h>'` -Linie in der Nähe der Spitze haben, wenn Sie die Funktionalität dieser Bibliothek verwenden möchten. Wie bei der Hardware-basierte 'SERIAL' Gerät, die Baudrate für 'SFTSER' gesetzt, um die Dropdown-Liste an seiner Unterseite mit - der gewählten Baudrate den Wert Ihres Programm geht auf die `'begin()'` Funktion entsprechen muß ordnungsgemäße Übertragung / Empfang. *Die serielle Kommunikation ist auf 8 Datenbits, 1 Stoppbit und kein Paritätsbit festgelegt.*

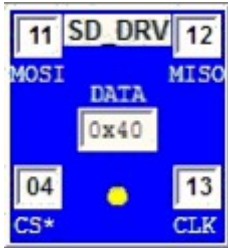


Wie beim hardwarebasierten `'Serial'` kann auch **ein größeres Fenster für TX- und RX-Einstellungen / -Anzeigen geöffnet werden, indem Sie auf das SFTSER-Gerät doppelklicken (oder mit der rechten Maustaste klicken)**.

Beachten Sie, dass im Gegensatz zur Hardware-Implementierung von `'Serial'` keine von internen ATmega-Interrupt-Vorgängen unterstützten **TX-** oder **RX-Puffer** bereitgestellt werden, sodass `'read()'` und `'write()'` blockieren (**dh**, Ihr Programm wird nicht fortgesetzt bis sie abgeschlossen sind).

SD-Laufwerk ('SD_DRV')

Dieses 'I / O' -Gerät ermöglicht bibliothekssoftwarevermittelte (aber **nicht** "bit-banged") Dateieingabe- und -ausgabevorgänge an den 'Uno' **SPI**- Pins (Sie können auswählen, welchen **CS*** -Pin Sie verwenden werden). Ihr Programm kann einfach '**#enthalten <SD.h>**' Zeile in der Nähe der Spitze, und Sie können '**<SD.h>**' Funktionen ODER direkt '**SdFile**' Funktionen selbst **aufrufen** .



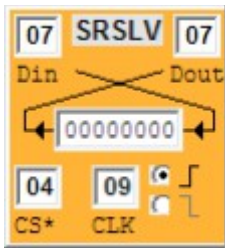
Ein **größeres Fenster mit Verzeichnissen und Dateien (und Inhalt) kann durch Doppelklicken (oder Rechtsklick) auf das 'SD_DRV' -Gerät geöffnet werden** . Alle Platteninhalt **von** einem **SD** - Unterverzeichnis in dem geladenen Programmverzeichnis **geladen** (falls vorhanden) bei '**SdVolume init()**' **und ist mit** demselben **SD** Unterverzeichnis auf Datei **gespiegelt** '**close()**', '**remove ()**' , und auf '**makeDir()**' und '**rmdir()**' .

Während SPI-Transfers blinkt eine gelbe LED und 'DATA' zeigt das letzte 'SD_DRV' **Response**- Byte. Alle SPI-Signale sind genau und können in einem **Wellenformfenster**

angezeigt werden .

Schieberegister Sklave ('SRSLV')

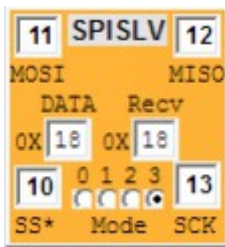
Dieses 'I / O' -Gerät emuliert ein einfaches Schieberegistergerät mit einem Aktiv-Niedrig- **SS*** -Pegel ("Sklave-Auswahl"), der den **Dout**- Ausgangspin **steuert** (wenn **SS*** hoch ist, wird **Dout** nicht angesteuert). Ihr Programm könnte die Funktionalität des integrierten SPI Arduino-Objekts und der Bibliothek verwenden. Alternativ können Sie auch eigene "Bit-Bang" -**Din** und **CLK**- Signale erstellen , um dieses Gerät zu steuern.



Das Gerät erkennt Flankenübergänge an seinem **CLK**- Eingang, die die Verschiebung seines Registers auslösen - die Polarität der abgetasteten **CLK**- Flanke kann unter Verwendung einer Funkknopfsteuerung gewählt werden. An jeder **CLK**- Flanke (der erfassten Polarität) erfasst das Register seinen **Din**- Pegel in die niedrigstwertige Bit- (LSB) -Position des Schieberegisters, während die verbleibenden Bits gleichzeitig um eine Position nach links in Richtung der MSB-Position verschoben werden. Immer wenn **SS*** niedrig ist, wird der aktuelle Wert in der MSB-Position des Schieberegisters auf **Dout** **gesetzt** .

Konfigurierbarer SPI-Sklave ('SPISLV')

Dieses 'I / O' -Gerät emuliert einen Modus-auswählbaren SPI-Sklave mit einem Aktiv-Niedrig- **SS*** -Pegel ("Sklave-Auswahl"), der den **MISO**- Ausgangspin steuert (wenn **SS*** hoch ist, wird **MISO** nicht angesteuert). Ihr Programm muss eine '**#include <SPI.h>**' -**Zeile haben** , wenn Sie die Funktionalität des integrierten SPI Arduino-Objekts und der Bibliothek verwenden möchten. Alternativ können Sie auch eigene "bit-banged" **MOSI**- und **SCK**- Signale erstellen , um dieses Gerät zu steuern .



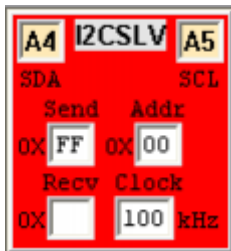
Das Gerät **erkennt Flankenübergänge** an seinem **CLK**- Eingang entsprechend dem ausgewählten Modus ('**MODE0**' , '**MODE1**' , '**MODE2**' oder '**MODE3**'), der entsprechend dem programmierten SPI-Modus Ihres Programms ausgewählt werden muss.

Durch einen Doppelklick (oder rechte Maustaste) auf dem Gerät können Sie ein größeren Begleiter Fenster öffnen, anstatt y ou ermöglicht in 32-Byte-Maximum - Puffer zu füllen (so wie SPI - Geräte zu emulieren , die automatisch ihre Daten zurück), und siehe

die letzten 32 empfangenen Bytes (alle als Hex-Paare). **Beachten Sie, dass** das nächste TX-Pufferbyte automatisch nur '**DATA**' entpackt wird, **nachdem** ein vollständiger '**SPI . transfer()**' abgeschlossen wurde!

Zweiadriger I2C-Sklave ('I2CSLV')

Dieses 'I / O' -Gerät emuliert nur ein Gerät im *Sklave-Modus*. Dem Gerät kann eine I2C-Busadresse unter Verwendung eines Zwei-Hex-Ziffern-Eintrags in seiner 'Addr' -eingabefeld zugewiesen werden (es wird nur auf I2C- Bus-Transaktionen antworten, die die zugewiesene Adresse umfassen). Das Gerät sendet und empfängt Daten auf seinem Open-Drain (nur Pulldown-Only) **SDA**- Pin und reagiert auf das Bustaktsignal an seinem Open-Drain (nur Pulldown-Only) **SCL**- Pin. Obwohl der 'Uno' der Busmaster ist, der für die Erzeugung des **SCL**- Signals verantwortlich ist, wird dieses Sklave-Gerät **SCL** auch während seiner niedrigen Phase niedrig ziehen, um (falls erforderlich) die Bus-Tieflaufzeit auf eine ihrer internen Geschwindigkeit entsprechende Zeit zu verlängern (das kann in seiner 'Clock' edit-box eingestellt werden).



Ihr Programm muss eine '`#include <Wire.h>`' Zeile haben, wenn Sie die Funktionalität der '`TwoWire`' Bibliothek verwenden möchten, um mit diesem Gerät zu interagieren. Alternativ können Sie Ihre eigenen Bit-gestoppten Daten- und Taktsignale zur Steuerung dieses Sklave-Geräts erstellen.

Ein einzelnes Byte für die Rückübertragung an den 'Uno' -Master kann in die 'Send' -Editbox gesetzt werden, und ein einzelnes (zuletzt empfangenes) Byte kann in seiner (schreibgeschützten) 'Recv' -Edit-Ansicht angezeigt werden. Box. **Beachten Sie, dass** der 'Send' -Editbox-Wert immer das **nächste** Byte für die Übertragung von diesem geräteinternen Datenpuffer angibt.

Wenn Sie auf das Gerät doppelklicken (oder mit der rechten Maustaste klicken), können Sie ein größeres Begleitfenster öffnen, in dem Sie stattdessen einen 32-Byte-FIFO-Puffer ausfüllen können (um TWI-Geräte mit einer solchen Funktionalität zu emulieren) und anzeigen (bis zu maximal 32) Bytes der zuletzt empfangenen Daten (als zweistellige Hexadezimalanzeige von 8 Bytes pro Zeile). Die Anzahl der Zeilen in diesen beiden Eingabefeldern entspricht der gewählten TWI-Puffergröße (die mit **Konfigurieren | Präferenzen** ausgewählt werden kann). Dies wurde als eine Option hinzugefügt, da die Arduino '`Wire.h`' -Bibliothek **fünf** solcher RAM-Puffer in ihrem Implementierungscode verwendet, was RAM-Speicher teuer ist. Durch das Editieren der Arduino Installation '`Wire.h`' Datei definierte konstante '`BUFFER_LENGTH`' ändern (und auch die Bearbeitung Begleiter '`utility / twi.h`' Datei TWI Pufferlänge zu ändern) sowohl stattdessen entweder 16 oder 8, könnte ein Benutzer den RAM-Speicher-Overhead des 'Uno' in einer gezielten Hardware-Implementierung deutlich reduzieren - UnoArduSim spiegelt diese reale Möglichkeit also über wider Konfigurieren | Präferenzen.

Schrittmotor ('STEPR')

Dieses 'I / O' -Gerät emuliert einen 6V bipolaren oder unipolaren Schrittmotor mit einem integrierten Treibercontroller, der **entweder durch zwei** (auf **P1**, **P2**) **oder vier** (auf **P1**, **P2**, **P3**, **P4**) Steuersignalen angesteuert wird. Die Anzahl der Schritte pro Umdrehung kann ebenfalls eingestellt werden. Sie können die '`Stepper.h`' -Funktionen '`setSpeed()`' und '`step()`' verwenden, um das 'STEPR' zu steuern. Alternativ *antwortet* 'STEPR' *auch auf* Ihre eigenen '`digitalWrite()`' "bit-banged" Treibersignale.



Der Motor wird sowohl mechanisch als auch elektrisch genau modelliert. Die Motortreiberspannung sinkt und variierende Reluktanz und Induktivität werden zusammen mit einem realistischen Trägheitsmoment in Bezug auf das Haltemoment modelliert. Die Motorrotorwicklung hat einen modellierten Widerstand von $R = 6 \text{ Ohm}$ und eine Induktivität von $L = 6 \text{ Milli-Henries}$, was eine elektrische Zeitkonstante von 1,0 Millisekunden erzeugt. Aufgrund der realistischen Modellierung werden Sie feststellen, dass sehr schmaler Steuerstift Impulse bemerken *nicht*, den Motor *bekommen* zu Schritt - sowohl aufgrund der endlichen Stromanstiegszeit, und die Wirkung der Rotorträgheit. Dies stimmt mit dem überein, was beim Fahren eines echten Schrittmotors von einem 'Uno' beobachtet wird, wobei natürlich ein geeigneter (**und erforderlicher**) Motortreiberchip zwischen den Motorkabeln und dem 'Uno'!

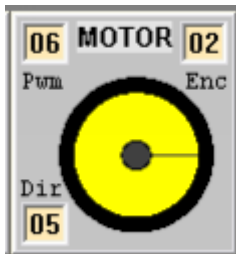
Ein unglücklicher Fehler im Arduino '`Stepper.h`' Bibliothekscode bedeutet, dass der Schrittmotor beim Zurücksetzen nicht in `Schrittposition 1` (von vier Schritten) ist. Um dies zu **umgehen**, sollte der Benutzer

'digitalWrite()' in seiner 'setup()' -Routine verwenden, um die **Steuerpin-** Ebenen auf die 'step (1)' -**Pege1** für 2-Pin (0,1) oder 4 zu initialisieren -pin (1,0,1,0) Kontrolle, und ermöglichen dem Motor 10 ms zu bewegen, um die 12-Uhr-Referenz erste gewünschte Motorposition.

Beachten Sie, dass die **Getriebeuntersetzung** aufgrund von Platzmangel **nicht direkt unterstützt wird**. Sie können sie jedoch in Ihrem Programm emulieren, indem Sie eine Modulo-N- **Zählervariable** implementieren und nur 'step()' aufrufen, wenn dieser Zähler auf 0 (für Untersetzung um den Faktor N).

Gleichstrom Motor ('MOTOR')

Dieses 'I / O' -Gerät emuliert einen 6-Volt-100: 1-Gleichstrommotor mit einem integrierten Treibercontroller, der durch ein Pulsweitenmodulationssignal (an seinem **Pwm-** Eingang) und einem Richtungssteuersignal (an seinem **Dir-** Eingang) **angesteuert wird**. Der Motor hat auch einen Radcodiererausgang, der seinen **Enc-** Ausgangsstift steuert. Sie können 'analogWrite()' **verwenden**, um den **Pwm-** Pin mit einer PWM-Signalfrequenz mit 490 Hz (auf Pins 3,9,10,11) oder 980 Hz (auf Pins 5,6) zwischen 0,0 und 1,0 zu **treiben** ('analogWrite()' Werte 0 bis 255). Alternativ *antwortet* 'MOTOR' *auch auf* Ihre eigenen 'digitalWrite()' "bit-banged" **Treibersignale**.



Der Motor wird sowohl mechanisch als auch elektrisch genau modelliert. Die Berücksichtigung von Motortreibertransistorspannungsabfällen und realistischem Leerlaufdrehmoment ergibt eine volle Geschwindigkeit von ungefähr 2 Umdrehungen pro Sekunde und ein Blockierdrehmoment von etwas über 5 kg-cm (das bei einem stetigen PWM-Tastverhältnis von 1,0 auftritt) mit a Gesamtmotor-plus-Last Trägheitsmoment von 2,5 kg-cm. Die Motorrotorwicklung hat einen modellierten Widerstand von $R = 2 \text{ Ohm}$ und eine Induktivität von $L = 300 \text{ Mikro-Henries}$, die eine elektrische Zeitkonstante von 150 Mikrosekunden erzeugt. Aufgrund der realistischen Modellierung werden Sie feststellen, dass sehr schmale PWM - Impulse bemerken *nicht*, den Motor *bekommen* zu drehen -

sowohl aufgrund der endlichen Stromanstiegszeit und dem erheblichen Aus-Zeit nach jedem schmalen Impulse. Diese kombinieren, um ein unzureichendes Rotormoment zu verursachen, um das getriebeartige federartige Zurückschlagen unter Haftreibung zu überwinden. Die Konsequenz ist, dass bei Verwendung von 'analogWrite()' ein Arbeitszyklus unter etwa 0,125 den Motor nicht zum **Rütteln** bringt - dies stimmt mit dem überein, was beim Fahren eines echten Getriebemotors von einem 'Uno' beobachtet wird, natürlich mit einem entsprechenden (**und benötigt**) Motortreibermodul zwischen dem Motor und dem 'Uno'!

Der emulierte Motorcodierer ist ein wellenmontierter optischer Unterbrechungssensor, der eine Wellenform mit 50% Tastverhältnis mit 8 vollständigen Hoch-Tief-Perioden pro Radumdrehung erzeugt (so kann Ihr Programm Raddrehungsänderungen mit einer Auflösung von 22,5 Grad erfassen).

ServoMotor ('SERVO')

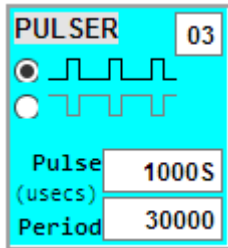
Sein 'I / O' Gerät emuliert einen positionsgesteuerten PWM-driven 6-Volt - Versorgung DC - Servomotor. Die mechanischen und elektrischen Modellierungsparameter für den Servobetrieb stimmen eng mit denen eines Standard-Servos HS-422 überein. Das Servo hat eine maximale Rotationsgeschwindigkeit von ungefähr 60 Grad in 180 ms. Wenn das Kontrollkästchen unten links aktiviert ist, wird das Servo zu einem Servo mit kontinuierlicher Rotation mit der gleichen maximalen Geschwindigkeit, aber jetzt stellt die PWM-Impulsbreite die Geschwindigkeit und nicht den Winkel ein



Ihr Programm muss eine '#include <Servo.h>' -Zeile haben, bevor Sie Ihre Servo-Instanz (en) deklarieren, *wenn Sie sich für die Verwendung der Servo-Bibliotheksfunktion entscheiden*, z. B. 'Servo.write()', 'Servo.writeMicroseconds()' Alternativ reagiert SERVO auch auf 'digitalWrite()' "bit-banged" Signale. Aufgrund der internen Implementierung von UnoArduSim sind Sie auf 5 'SERVO' -Geräte beschränkt.

Digitaler Impulsgeber ('PULSER')

Dieses 'I / O' -Gerät emuliert einen einfachen digitalen Impulswellenformgenerator, der ein periodisches Signal erzeugt, das an jeden beliebigen 'Uno' -Pin angelegt werden kann.

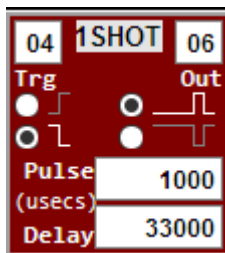


Die Periode und die Pulsbreiten (in Mikrosekunden) können unter Verwendung von Editierboxen eingestellt werden - die minimal erlaubte Periode beträgt 50 Mikrosekunden und die minimale Pulsbreite beträgt 10 Mikrosekunden. Die Polarität kann ebenfalls gewählt werden: entweder positive Vorderflankenimpulse (0 bis 5 V) oder negative Vorderflankenimpulse (5 V bis 0 V).

'Pulse' und **'Period'** -Werte sind skalierbar vom Haupt- **Tool-Bar** 'I / O ____ S' -Faktor-Schieberegler-Steuerelement, indem Sie eines (oder beide) mit dem Buchstaben 'S' (oder 's') ersetzen.

Generator Ein-Schuss ('1SHOT')

Dieses 'I / O' -Gerät emuliert einen digitalen Generator Ein-Schuss, der an seinem **'Out'** einen Puls mit der gewählten Polarität und Pulsbreite erzeugen kann. Pin, der nach einer bestimmten Verzögerung von einer Triggerflanke auftritt, die an seinem **Trg** (Trigger) -Eingangspin empfangen wird. Sobald die spezifizierte Triggerflanke empfangen wird, beginnt der Zeitablauf und dann wird ein neuer Triggerimpuls nicht erkannt, bis der **'Out'** -Puls erzeugt wurde (und vollständig abgeschlossen ist).

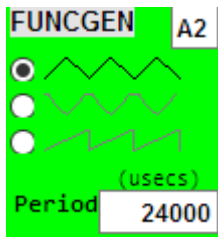


Eine mögliche Verwendung dieser Vorrichtung besteht darin, Ultraschall-Entfernungssensoren zu simulieren, die einen Entfernungsimpuls als Antwort auf einen Auslöseimpuls erzeugen. Es kann auch überall dort verwendet werden, wo Sie ein Pin-Eingangssignal erzeugen möchten, das (nach Ihrer gewählten Verzögerung) mit einem von Ihrem Programm erzeugten Pin-Ausgangssignal synchronisiert ist.

'Pulse' und **'Delay'** Werte sind skalierbar vom Hauptfenster **Tool-Bar** 'I / O ____ S' Skalierungsfaktor-Schieberegler-Kontrolle durch Suffix von einem (oder beiden) mit dem Buchstaben 'S' (oder 's').

Analoger Funktionsgenerator ('FUNCGEN')

Dieses 'I / O' -Gerät emuliert einen einfachen analogen Signalgenerator, der ein periodisches Signal erzeugt, das an jeden beliebigen 'Uno' -Pin angelegt werden kann.



Die Dauer (in Mikrosekunden) kann mit der eingabefeld eingestellt werden - die minimal zulässige Dauer beträgt 100 Mikrosekunden. Die Wellenform, die erzeugt wird, kann sinusförmig, dreieckig oder sägezahnförmig gewählt werden (um eine Rechteckwelle zu erzeugen, verwenden Sie stattdessen einen 'PULSER'). Bei kleineren Perioden werden weniger Abtastungen pro Zyklus verwendet, um die erzeugte Wellenform zu modellieren (nur 4 Abtastungen pro Zyklus bei Periode = 100 usecs).

Der **'Period'** -Wert ist skalierbar vom Hauptfenster **Tool-Bar** 'I / O ____ S' Skalierungsfaktor-Schiebereglersteuerung, indem er an den Buchstaben 'S' (oder 's') angehängt wird.

Piezelektrischer Lautsprecher ('PIEZO')



Mit diesem Gerät können Sie Signale an jedem beliebigen 'Uno' -Pin "abhören". Dies kann eine nützliche Ergänzung zu LEDs zum Debuggen Ihres Programmbetriebs sein. Sie können auch ein wenig Spaß beim Abspielen von Klingeltönen durch entsprechende **Aufrufe** von `'tone()'` und `'delay()'` **haben** (obwohl es keine Filterung der rechteckigen Wellenform gibt, so dass Sie keine "reinen" Töne hören).

Sie können auch ein angeschlossenes 'PULSER' - oder 'FUNCGEN' -Gerät abhören, indem Sie ein 'PIEZO' mit dem Pin verbinden, den das Gerät steuert.

Druckknopf ('PUSH')



Dieses 'I / O' -Gerät emuliert einen normalerweise offenen **kurzzeitigen oder verriegelnden** Einpol- (SPST) -Taster mit einem 10 k-Ohm-Pull-Up- (oder Pull-Down-) Widerstand. Wenn für das Gerät eine Übergangsflanke gewählt wird, werden die Kontakte des Tasters zwischen dem Geräte-Pin und +5 V mit einem 10 k-Ohm-Pulldown-Anschluss an Masse verdrahtet. Wenn für das Gerät ein Übergang mit fallender Flanke gewählt wird, werden die Kontakte des Tasters zwischen dem Pin des Geräts und Masse mit einem 10 k-Ohm-Pull-up-Anschluss mit +5 V verdrahtet.

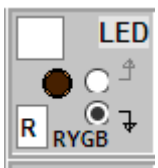
Durch Klicken mit der linken Maustaste auf die Schaltfläche oder durch Drücken einer beliebigen Taste schließen Sie den Drucktastenkontakt. Im **momentanen** Modus bleibt es so lange geschlossen, wie Sie die Maustaste gedrückt halten, und im **Latch-** Modus (aktiviert durch Klicken auf die Schaltfläche **'latch'**) bleibt es geschlossen (und eine andere Farbe), bis Sie die Taste erneut drücken. Das Kontaktprellen (für 1 Millisekunde) wird jedes Mal erzeugt, wenn Sie die Leertaste drücken, um den Druckknopf zu drücken.

Schiebeschalterwiderstand ('R = 1K')



Mit diesem Gerät kann der Benutzer an einen 'Uno' -Stift entweder einen 1 k-Ohm-hochziehen-Widerstand an + 5V oder einen 1 k-Ohm- herunterziehen-Widerstand an Masse anschließen. Auf diese Weise können Sie elektrische Lasten simulieren, die zu einem echten Hardwaregerät hinzugefügt werden. Durch Linksklick auf dem Schiebeschalter **Körper** können Wählen Sie hochziehen schalten oder herunterziehen - Auswahl. Wenn Sie eines oder mehrere dieser Geräte verwenden, können Sie einen einzelnen (oder mehrere) "Bit" -Code für Ihr Programm festlegen, das gelesen und darauf reagiert wird.

Farbige LED ('LED')



Sie können eine LED zwischen dem gewählten 'Uno' -Pin (durch einen eingebauten, versteckten 1 k-Ohm-Strombegrenzungswiderstand) entweder an Masse oder an + 5V anschließen - dies gibt Ihnen die Wahl, ob die LED aufleuchtet connected 'Uno' pin ist **'HIGH'** oder stattdessen, wenn es **'LOW'** ist . Die LED-Farbe kann entweder rot ('R'), gelb ('Y'), grün ('G') oder blau ('B') über die eingabefeld gewählt werden.





Analoger Schieberegler








Ein slidergesteuertes 0-5V Potentiometer kann an jeden beliebigen 'Uno' Pin angeschlossen werden, um einen statischen (oder langsam ändernden) analogen Spannungspegel zu erzeugen, der von `'analogRead()'` als Wert von 0 bis 1023 gelesen wird . Ziehen Sie den analogen Schieberegler mit der Maus oder klicken Sie, um zu springen.

Menüs









Datei:

 Lade INO oder PDE Programm (Strg-L)	Ermöglicht dem Benutzer, eine Programmdatei mit der ausgewählten Erweiterung auszuwählen. Das Programm wird sofort analysiert
Editieren / Ansehen (Strg-E)	Öffnet das geladene Programm zum Anzeigen / Editieren.
 Speichern	Speichern Sie den bearbeiteten Programminhalt zurück in die ursprüngliche Programmdatei.
Speichern Als	Speichern Sie den bearbeiteten Programminhalt unter einem anderen Dateinamen.
 Nächste ('#include')	Bringt den CodeBereich vor , um die nächste ' #include ' -Datei anzuzeigen
 Vorherige	Gibt die CodeBereichanzeige an die vorherige Datei zurück
Ausgang	Beendet UnoArduSim, nachdem der Benutzer daran erinnert wurde, geänderte Dateien zu speichern.

Finden:

 Nächste Funktion/Var	Springe zur nächsten Funktion im CodeBereich (wenn sie den aktiven Fokus hat) oder zur nächsten Variablen im VariablenBereich (wenn sie stattdessen den aktiven Fokus hat).
 Vorherige Funktion/Var	Springe zur vorherigen Funktion im CodeBereich (wenn es den aktiven Fokus hat) oder zur vorherigen Variablen im VariablenBereich (wenn es stattdessen den aktiven Fokus hat).
 Setze Suchtext (Strg-F)	Aktivieren Sie das Bearbeitungsfeld Werkzeugleiste Suchen, um den Text zu definieren, nach dem als nächstes gesucht werden soll (und fügt das erste Wort aus der aktuell markierten Zeile im CodeBereich oder VariablenBereich hinzu, wenn einer davon den Fokus hat).
 Finde den Nächsten Text	Springt zum nächsten Text-Vorkommen im CodeBereich (wenn es den aktiven Fokus hat) oder zum nächsten Text-Vorkommen im VariablenBereich (wenn es stattdessen den aktiven Fokus hat).
 Vorherigen Text Finden	Zum vorherigen Text-Vorkommen im CodeBereich springen (wenn es den aktiven Fokus hat) oder zum vorherigen Text-Vorkommen im VariablenBereich (wenn es stattdessen den aktiven Fokus hat).

Ausführen:

 Schritt Hinein (F4)	Die Ausführung wird um eine Anweisung vorwärts oder <i>in eine aufgerufene Funktion fortgesetzt</i> .
 Schritt Über (F5)	Die Ausführung wird um eine Anweisung vorwärts oder <i>um einen vollständigen Funktionsaufruf</i> .
 Schritt Aus (F6)	Erweitert die Ausführung um <i>gerade genug, um die aktuelle Funktion zu verlassen</i> .
 Ausführen Dort (F7)	Führt das Programm aus <i>und hält an der gewünschten Programmzeile an</i> - Sie müssen zuerst auf eine gewünschte Programmzeile klicken, bevor Sie Ausführen-Dort verwenden.
 Ausführen Bis(F8)	Führt das Programm so lange aus, bis die Variable mit der aktuellen Hervorhebung im VariablenBereich aufschreibt (klicken Sie auf eine Variable , um die anfängliche Hervorhebung festzulegen).
 Ausführen (F9)	Führt das Programm aus.
 Halt (F10)	Stoppt die Programmausführung (<i>und friert die Zeit ein</i>).
 Zurücksetzen	Setzt das Programm zurück (alle Wertvariablen werden auf den Wert 0 zurückgesetzt und alle Zeigervariablen werden auf 0x0000 zurückgesetzt).
Animieren	Schreitet automatisch fortlaufende Programmzeilen <i>mit zusätzlicher künstlicher Verzögerung</i> und Hervorhebung der aktuellen Codezeile. Echtzeitbetrieb und Sounds sind verloren.
Zeitlupe	Verlangsamt die Zeit um den Faktor 10.

Optionen:

<u>Schritt Über Strukturen / Operatoren</u>	Fliege während eines Schrittes direkt durch Konstruktoren, Destruktoren und Operator-Überladungsfunktionen (dh es wird nicht innerhalb dieser Funktionen gestoppt).
<u>Register-Zuweisungsmodellierung</u>	Weisen Sie Funktion Locals zu, ATmega-Registern statt zu dem Stapel freizugeben (verursacht etwas reduzierte RAM-Auslastung).
<u>Fehler bei Nicht initialisiert</u>	Wird als Parse-Fehler überall dort angezeigt, wo Ihr Programm versucht, eine Variable zu verwenden, ohne zuvor ihren Wert (oder mindestens einen Wert in einem Array) initialisiert zu haben.
<u>Hinzugefügt 'loop()' Verzögerung</u>	Addiert 200 Mikrosekunden Verzögerung jedes Mal, wenn 'loop()' aufgerufen wird (falls keine weiteren Programmaufrufe an 'delay()' irgendwo vorhanden sind) - nützlich, um zu vermeiden, zu weit hinter Echtzeit zu fallen.

Konfigurieren:

<u>'I / O' Geräte</u>	Öffnet einen Dialog, in dem der Benutzer die gewünschten Typen und Nummern der gewünschten 'I / O' -Geräte auswählen kann. In diesem Dialogfeld können Sie auch 'I / O' Geräte in eine Textdatei speichern und / oder 'I / O' Geräte aus einer zuvor gespeicherten (oder bearbeiteten) Textdatei laden (einschließlich aller PIN-Anschlüsse und anklickbaren Einstellungen und Eingaben Werte).
<u>Präferenz</u>	Öffnet einen Dialog, in dem der Benutzer Präferenz wie das automatische Einrücken von Quellprogrammzeilen festlegen kann. Expertensyntax, Schriftartenwahl, größere Schriftgröße, Erzwingen von Array-Grenzen, Erlauben von logischen Operatorwörtern, ProgrammHerunterladen, Auswahl von 'Uno' Leiterplatte-Version und TWI-Pufferlänge (für I2C-Geräte).

VarAktualisieren:

<u>Erlaube Automatisches(-) Kontraktion</u>	Lassen Sie UnoArduSim angezeigte erweiterte Arrays / Objekte ausblenden, wenn sie in Echtzeit zurückfallen.
<u>Erlaube Reduktion</u>	Lassen Sie im VariablenBereich eine geringere Häufigkeit von Bildschirmaktualisierungen zu, um Flimmern zu vermeiden oder die CPU-Last zu reduzieren, wenn Sie hinter der Echtzeit zurückbleiben - dann werden die angezeigten Werte nur in regelmäßigen Abständen, aber auch bei angehaltenem Programm aktualisiert .
<u>Minimal</u>	Aktualisieren Sie die VariablenBereich - Anzeige nur 4 Mal pro Sekunde.
<u>Änderungen Hervorheben</u>	Markieren Sie den zuletzt geänderten Variablenwert (kann Verlangsamung verursachen).

Fenster:

<u>'Serial' Monitor</u>	Verbinden Sie ein serielles E / A-Gerät mit den Pins 0 und 1 (falls nicht vorhanden), und ziehen Sie ein größeres 'Serial' -Monitor TX / RX-Textfenster nach oben.
<u>Alles Wiederherstellen</u>	Stellen Sie alle minimierten untergeordneten Fenster wieder her.
<u>Pin DigitalWellenformen</u>	Stellen Sie ein minimiertes Pin DigitalWellenformen Fenster wieder her.
<u>Pin AnalogWellenform</u>	Stellen Sie ein minimiertes Pin AnalogWellenform Fenster wieder her.

Hilfe:

<u>Schnellhilfe-Datei</u>	Öffnet die UnoArduSim_QuickHelp PDF-Datei.
<u>Vollständige Hilfedatei</u>	Öffnet die UnoArduSim_FullHelp PDF-Datei.
<u>Fehlerbehebung</u>	Wichtige Fehlerkorrekturen seit der letzten Version
<u>Änderung / Verbesserungen</u>	Wichtige Änderungen und Verbesserungen seit der letzten Version
<u>Über</u>	Zeigt Version, Copyright an.

Modellieren

'Uno' Leiterplatte und 'I / O' Geräte

Die 'Uno' und attached 'I / O' -Geräte sind alle genau elektrisch modelliert, und Sie werden zu Hause eine gute Vorstellung davon bekommen, wie sich Ihre Programme mit der tatsächlichen Hardware verhalten, und alle elektrischen Pin-Konflikte werden markiert.

Zeitliche Koordinierung

UnoArduSim wird auf einem PC oder Tablet schnell genug ausgeführt, so dass Programmaktionen *in der Regel* in Echtzeit modelliert werden können, **aber nur, wenn Ihr Programm** mindestens einige kleine '`delay()`' -Anrufe oder andere Anrufe enthält, die natürlich **beibehalten** werden es synchronisiert sich in Echtzeit (siehe unten).

Um dies zu erreichen, verwendet UnoArduSim eine Windows-Callback-Timer-Funktion, die es ermöglicht, die Echtzeitverfolgung genau zu verfolgen. Die Ausführung einer Anzahl von Programmanweisungen wird während eines **Zeitschlitzes** simuliert, und Befehle, die eine längere Ausführung erfordern (wie Aufrufe an '`delay()`'), müssen möglicherweise mehrere Zeitgeberscheiben verwenden. Jede Iteration der Callback-Timer-Funktion korrigiert die Systemzeit unter Verwendung des Systemhardwaretakts, so dass die Programmausführung ständig angepasst wird, um mit der Echtzeit-Synchronisation Schritt zu halten. *Die einzige Zeit, in der die Ausführungsrate hinter der Echtzeit zurückbleiben muss*, ist, wenn der Benutzer enge Schleifen **ohne zusätzliche Verzögerung erstellt** hat oder 'I / O' -Geräte für den Betrieb mit sehr hohen 'I / O' -Gerätefrequenzen (und / oder Baud) konfiguriert sind Rate), die eine übermäßige Anzahl von Änderungsereignissen auf Pin-Ebene und die damit verbundene Verarbeitungsüberlastung erzeugen würde. UnoArduSim bewältigt diese Überlastung, indem einige Timer-Intervalle zur Kompensation übersprungen werden, was wiederum den Programmfortschritt **unter Echtzeit** verlangsamt .

Darüber hinaus können Programme, bei denen große Arrays angezeigt werden oder die wiederum enge Schleifen **ohne zusätzliche Verzögerung aufweisen**, eine hohe Funktionsaufrufhäufigkeit verursachen und eine hohe Aktualisierungslast der **VariablenBereich**- Anzeige erzeugen, die dazu führt, dass sie hinter Echtzeit zurückbleibt - dies kann mit **Zulassen** umgangen werden **Reduktion** , oder wenn wirklich benötigt, **Minimal**, aus dem Menü **VarAktualisieren** ,

Eine genaue Modellierung der Ausführungszeit von **weniger als einer** Millisekunde für jede Programmanweisung oder Operation **wird nicht durchgeführt** - für Simulationszwecke wurden nur sehr grobe Schätzungen angenommen. Das zeitliche koordinierung von '`delay()`' und '`delayMicroseconds()`' -Funktionen sowie die Funktionen '`millis()`' und '`micros()`' sind jedoch alle genau und **solange Sie mindestens eine der Verzögerungsfunktionen verwenden** in einer Schleife irgendwo in Ihrem Programm, **oder** Sie verwenden eine Funktion, die sich natürlich an Echtzeitbetrieb bindet (wie '`print()`', die an die gewählte Baudrate gebunden ist), dann ist die simulierte Leistung Ihres Programms sehr nahe in Echtzeit (wiederum ohne eklatante überhöhte hochfrequente PIN-Level-Änderungsereignisse oder exzessive vom Benutzer erlaubte Variablen-Updates, die dies verlangsamen könnten).

Um den Effekt individueller Programmanweisungen während des *Laufens zu sehen* , kann es wünschenswert sein, die Dinge verlangsamen zu können. Im Menü **Execute** kann vom Benutzer ein Zeitverzögerungsfaktor von 10 eingestellt werden .

'I / O' Gerätezeitsteuerung

Diese virtuellen Geräte erhalten Echtzeit-Signalisierung von Änderungen, die an ihren Eingangspins auftreten, und erzeugen entsprechende Ausgangssignale an ihren Ausgangspins, die dann von 'Uno' abgetastet werden können - sie sind daher von Natur aus mit der Programmausführung synchronisiert. Internes 'I / O' -Gerätezeitliche koordinierung wird vom Benutzer festgelegt (z. B. durch Baudratenauswahl oder Taktfrequenz) und Simulatorereignisse werden eingerichtet, um den internen Echtzeitbetrieb zu verfolgen.

Geräusche

Jedes 'PIEZO' -Gerät erzeugt einen Klang entsprechend den Änderungen des elektrischen Pegels, die an dem angeschlossenen Stift auftreten, unabhängig von der Quelle solcher Änderungen. Um die Sounds für die Programmausführung synchronisiert zu halten, startet und stoppt UnoArduSim die Wiedergabe eines zugehörigen Soundpuffers, wenn die Ausführung gestartet / angehalten wird.

Ab Version 2.0 wurde der Sound jetzt so modifiziert, dass er die Qt-Audio-API verwendet. Leider unterstützt QAudioOutput 'class' nicht, den Sound-Puffer zu loopen, um Sound-Samples zu vermeiden (wie bei längeren OS-Windowing-Verzögerungen). Um die große Mehrheit von lästigen Soundklicks und Soundunterbrechungen während OS-Verzögerungen zu vermeiden, wird der Sound daher gemäß der folgenden Regel stummgeschaltet:

Der Ton ist stummgeschaltet, solange UnoArduSim nicht das "aktive" Fenster ist (außer wenn ein neues untergeordnetes Fenster gerade erstellt und aktiviert wurde), **und auch** wenn UnoArduSim das "aktive" Hauptfenster ist, aber der Mauszeiger **außerhalb** seiner Hauptansicht ist Fenster-Client-Bereich.

Beachten Sie, dass dies bedeutet, dass der Ton vorübergehend als König stummgeschaltet **wird**, wenn der Mauszeiger über einem **untergeordneten Fenster** schwebt, **und stummgeschaltet wird**, wenn auf das **untergeordnete Fenster** geklickt wird, um es zu aktivieren (**bis das nain UnoArduSim Fenster erneut angeklickt wird, um ir wieder zu aktivieren**) .

Der Ton kann immer deaktiviert werden, indem Sie auf ianywhere im Client-Bereich des UoArduSim-Hauptfensters klicken.

Aufgrund der Pufferung hat das Signal eine Echtzeitverzögerung von bis zu 250 Millisekunden von der entsprechenden Ereigniszeit auf dem Pin des angeschlossenen 'PIEZO'.

Einschränkungen und nicht unterstützte Elemente

Enthaltene Dateien

A '<>' - geklammert '#enthält' von '<Servo.h>', '<Draht.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' und '<SD. h>' wird unterstützt, aber diese werden nur emuliert - die eigentlichen Dateien werden nicht gesucht; Stattdessen ist ihre Funktionalität direkt in UnoArduSim integriert und gilt für die fest installierte Arduino-Version.

Jedes quoted '#include' (zum Beispiel von "supp.ino", "myutil.cpp" oder "mylib.h") wird unterstützt, aber alle diese Dateien müssen **sich in der dasselbe Verzeichnis als die übergeordnete Programmdatei**, die enthält ihre '#include' (es gibt keine Suche in anderen Verzeichnissen). Das '#include' -Feature kann nützlich sein, um die Menge des im **CodeBereich** angezeigten Programmcodes zu jedem Zeitpunkt zu minimieren. Header-Dateien mit '#include' (dh solche mit der Erweiterung ".h") werden zusätzlich dazu führen, dass der Simulator versucht, die gleichnamige Datei mit der Erweiterung ".cpp" zu verwenden (falls sie auch im Verzeichnis des Elternteils existiert) Programm).

Dynamische Speicherzuordnungen und RAM

Betreiber 'new' und 'delete' werden unterstützt, ebenso wie native Arduino 'String' -Objekte, **aber keine direkten Aufrufe an** 'malloc()', 'realloc()' und 'free()' , auf die sich diese verlassen.

Übermäßiger RAM-Gebrauch für Variablendeklarationen wird zur Parse-Zeit gekennzeichnet, und RAM-Speicherüberlauf wird während der Programmausführung markiert. Ein Element in meny **Options** ermöglicht es

Ihnen, die normale ATmega-Registerzuordnung zu emulieren, wie dies der AVR-Compiler vornimmt , oder ein alternatives Kompilierungsschema zu modellieren, das nur den Stack verwendet (als Sicherheitsoption, falls ein Fehler in meiner Registerzuweisung auftaucht) Modellieren). Wenn Sie einen Zeiger verwenden, um den Stack-Inhalt zu betrachten, sollte er genau wiedergeben, was in einer tatsächlichen Hardware-Implementierung erscheinen würde.

'Flash' Speichervuordnungen

'Flash' datenbank 'byte' , 'int' und 'float' variables / arrays und ihre entsprechenden Lesezugriffsfunktionen werden unterstutzt. Irgendein 'F()' Funktionsaufruf ("Flash'-macro") beliebiger ZeichenfolgenLiteralen wird unterstutzt, aber die einzigen 'Flash'-Datenbank-Zeichenfolgen direkten Zugriffsfunktionen sind 'strcpy_P()' und 'memcpy_P()' , um andere Funktionen zu verwenden Sie mussen zunachst die 'Flash' Zeichenfolge in einen normalen RAM kopieren 'String' Variable, und dann mit dem RAM 'String' work. Wenn Sie das 'PROGMEM' variable-modifier-Schlsselwort verwenden, muss es **vor** dem Variablenamen erscheinen, und diese Variable **muss auch** als 'const' deklariert werden .

'String' Variablen

Die native 'String' -Bibliothek wird fast vollstndig mit einigen wenigen (und kleineren) Ausnahmen unterstutzt.

Die 'String' Operatoren sind +, + =, <, <=,>,> =, ==,! = und []. **Beachten Sie Folgendes :** 'concat()' verwendet ein **einzelnes** Argument, bei dem es sich um den 'String' oder 'char' oder 'int' handelt , der an das ursprngliche 'String' -Objekt angehtngt werden soll, **nicht an** zwei Argumente, wie flschlicherweise in der Arduino-Referenz angegeben Webseiten).

Arduino-Bibliotheken

Nur 'Stepper.h' , 'SD.h' , 'Servo.h' , 'SoftwareSerial.h' , 'SPI.h' , 'Wire.h' und 'EEPROM.h' fr die **Version Arduino V1.6.6 werden** derzeit in UnoArduSim unterstutzt. Der Versuch, '#einzuschlieen,' Die ".cpp" und ".h" Dateien von anderen noch nicht unterstutzten Bibliotheken **funktionieren nicht**, da sie Low-Level-Assembler-Anweisungen und nicht unterstutzte Direktiven und nicht erkannte Dateien enthalten!

Zeiger

Zeiger auf einfache Typen, Arrays oder Objekte werden unterstutzt. Ein Zeiger kann einem Array des gleichen Typs (z. B. `iptr = intarray`), aber dann wrde es *keine nachfolgenden Array-Grenzen geben, die auf einen Ausdruck wie `iptr [index]` prfen wrden* .

Funktionen knnen Zeiger oder 'const' -Zeiger zurckgeben , aber jede nachfolgende Ebene von 'const' auf dem zurckgegebenen Zeiger wird ignoriert.

Es gibt **keine Unterstutzung** fr Funktionsaufrufe, die durch von **Benutzern deklarierte Funktionszeiger ausgefuhrt werden** .

Objekte 'class' und 'struct'

Obwohl Poly-Morphism und Vererbung (in beliebiger Tiefe) unterstutzt wird, kann eine 'class' oder 'struct' nur so definiert werden, dass sie hchstens **eine** base 'class' hat (dh **Mehrfachvererbung** wird nicht unterstutzt). Base 'class' Konstruktorinitialisierungsaufrufe (ber die Doppelpunktnotation) in Konstruktordeklarationszeilen werden unterstutzt, nicht jedoch **Mitgliederinitialisierungen, die dieselbe Doppelpunktnotation verwenden**. Dies bedeutet, dass Objekte, die 'const' non-'static' -Variablen

oder Variablen vom Referenztyp enthalten, nicht unterstützt werden (dies ist nur mit bestimmten Mitglieder-Initialisierungen für die Bauzeit möglich).

Operatorüberladungen für Kopierzuweisungen werden zusammen mit move-constructors und move-assignments unterstützt, aber benutzerdefinierte Objektkonvertierungsfunktionen ("Cast-like") werden nicht unterstützt.

Umfang

Es gibt keine Unterstützung für das Schlüsselwort **'using'** oder für Namespaces oder für **'file'** scope. Alle nicht-lokalen Deklarationen werden nach Implementierung als global angenommen.

Any **'typedef'**, **'struct'**, oder **'class'** definition (Das kann für zukünftige Deklarationen verwendet werden), muss **globaler** Geltungsbereich sein (**lokale** Definitionen solcher Elemente innerhalb einer Funktion werden nicht unterstützt).

Qualifikanten **'unsigned'** , **'const'** , **'volatile'** , **'static'**

Das Präfix **'unsigned'** funktioniert in allen normalen rechtlichen Kontexten. Das Schlüsselwort **'const'** muss , wenn es verwendet wird, dem Variablennamen oder Funktionsnamen oder **'typedef'** name **vorausgehen**, der deklariert wird. Wenn Sie ihn nach dem Namen **einfügen** , wird ein Parse-Fehler verursacht. Bei Funktionsdeklarationen können nur Zeiger zurückgebende Funktionen **'const'** in ihrer Deklaration enthalten.

Alle UnoArduSim-Variablen sind **'volatile'** durch Implementierung, daher wird das **'volatile'** -Schlüsselwort einfach in allen Variablendeklarationen ignoriert. Funktionen dürfen nicht **'volatile'** deklariert werden , ebenso wenig wie Funktionsaufrufargumente.

Das **'static'** -Schlüsselwort ist für normale Variablen und für **Objektelemente und Elementfunktionen** zulässig, ist jedoch explizit für Objektinstanzen (**'class'** / **'struct'**), für Nichtmitgliedsfunktionen und für alle Funktionsargumente nicht zulässig .

Compiler-Richtlinien

'#include' und regular **'#define'** werden beide unterstützt, aber **nicht macro** **'#define'** . Das **'#Pragma'** Direktive und bedingte Inclusion Direktiven (**'#ifdef'** , **'#ifndef'** , **'#if'** , **'#endif'** , **'#else'** und **'#elif'**) werden ebenfalls **nicht unterstützt** . Die **'#-Zeile'** , **'#Fehler'** und vordefinierte Makros (wie **'_LINE_'** , **'_FILE_'** , **'_DATE_'** und **'_TIME_'**) werden ebenfalls **nicht unterstützt**.

Arduino-Sprachelemente

Alle nativen Arduino-Sprachelemente werden mit Ausnahme der dubiosen **'goto'** -Anweisung **unterstützt** (die einzige vernünftige Verwendung, die ich mir **vorstellen** kann, wäre ein Sprung (zu einer Bail-Out- und Safe-Shutdown-Endlosschleife) im Falle von eine Fehlerbedingung, die Ihr Programm nicht anders behandeln kann).

C / C ++ - Sprachelemente

Bitsparende "Bitfeldqualifikatoren" für Mitglieder in Strukturdefinitionen werden **nicht unterstützt** .

'union' wird **nicht unterstützt**.

Der oddball "Kommaoperator" wird **nicht unterstützt** (Sie können also nicht mehrere durch Kommas getrennte Ausdrücke ausführen, wenn normalerweise nur ein einzelner Ausdruck erwartet wird, z. B. in `'while()'` und `'for (;;)'` Konstrukten).

Funktionsvorlagen

Benutzerdefinierte Funktionen, die das Schlüsselwort "template" verwenden, um die Annahme von Argumenten vom Typ "generic" zu ermöglichen, werden **nicht unterstützt**.

Echtzeit-Emulation

Wie oben erwähnt, sind die Ausführungszeiten der vielen verschiedenen individuellen Arduino-Programmanweisungen **nicht** genau modelliert, so dass Ihr Programm eine Art dominierender `'delay()'` -Anweisung benötigt (mindestens einmal, um in Echtzeit zu laufen) per `'loop()'`, oder eine Anweisung, die natürlich mit Echtzeit-Änderungen auf PIN-Ebene synchronisiert wird (z. B. `'pulseIn()'`, `'shiftIn()'`, `'Serial.read()'`, `'Serial). print()'`, `'Serial.flush()'` usw.).

Weitere Informationen zu Einschränkungen finden Sie unter Modellierung und **Geräusche** oben.

Versionshinweise

Fehlerbehebung

V2.0.1- Jan. 2018

- 1) In nicht-englischen Gebietsschemas wurde `'en'` fälschlicherweise so angezeigt, wie es in den **Präferenzen** ausgewählt wurde, was dazu führte, dass die englische Sprache umständlich wurde (Abwahl und erneute Auswahl erforderlich).
- 2) Es war dem Benutzer möglich, einen Device-Pin-Eingabefeld-Wert in einem unvollständigen Zustand (wie `'A_'`) zu belassen und die 'DATA'-Bits eines 'SRSlave' unvollständig zu lassen.
- 3) Die maximale Anzahl der Analoge Schieberegleren wurde auf 4 begrenzt (korrigiert jetzt auf 6).
- 4) UnoArduSim besteht nicht länger darauf, dass `'='` in einer Array-Aggregat-Initialisierung erscheint.
- 5) UnoArduSim hat darauf bestanden, dass das Argument `'inverted_logic'` an `'SoftwareSerial()'` übergeben wird.
- 6) Bit-Shift-Operationen erlauben nun Verschiebungen länger als die Größe der verschobenen Variablen.

V2.0- Dez. 2017

- 1) Alle Funktionen, die als `'unsigned'` deklariert wurden, hatten trotzdem Werte zurückgegeben, als wären sie `'signed'`. Dies hatte keine Auswirkungen, wenn der `'return'`-Wert einem `'unsigned'` zugewiesen wurde. Variable, hätte aber eine falsch negative Interpretation verursacht, wenn `MSB == 1` wäre, und sie wurde dann einer `'signed'`-Variable zugewiesen oder in einer Ungleichung getestet.
- 2) Analoge Schieberegleren erreichten nur einen maximalen `'analogRead()'`-Wert von 1022, nicht den korrekten 1023.
- 3) Ein Fehler, der versehentlich in V1.7.0 in Logik eingeführt wurde, um die Handhabung des SPI-System-SCK-Pins zu beschleunigen, führte dazu, dass SPI-Transfers für `'SPI_MODE1'` und `'SPI_MODE3'` nach dem ersten übertragenen Byte fehlschlügen (ein zusätzlicher SCK-Übergang folgte jedem Byte). Auch Aktualisierungen an einer 'SPISLV' edit 'DATA' Box für übertragene Bytes wurden verzögert.
- 4) Das farbige LED-Gerät hat 'B' (für Blau) nicht als Farboption aufgelistet (obwohl es akzeptiert wurde).
- 5) Einstellungen für 'SPISLV' und 'I2CSLV' Geräte wurden nicht in der Datei Benutzer **'I / O' Geräte** gespeichert.
- 6) Das Kopieren von `'Servo'`-Instanzen ist aufgrund einer fehlerhaften Implementierung von `'Servo :: Servo (Servo & tocopy)'` copy-constructor fehlgeschlagen.
- 7) Werte außerhalb des Bereichs `'Servo.writeMicroseconds()'` wurden ordnungsgemäß als Fehler erkannt, die angegebenen Grenzwerte für den Text der Fehlermeldung waren jedoch falsch.
- 8) Eine zulässige Baudrate von 115200 wurde nicht akzeptiert, wenn sie aus einer **'I / O' Geräte** Textdatei geladen wurde.
- 9) Elektrische Stiftkonflikte, die durch ein angeschlossenes Analoge Schieberegler-Gerät verursacht wurden, wurden nicht immer erkannt.
- 10) In seltenen Fällen kann das Übergeben eines fehlerhaften Zeichenfolgenzeigers (mit der fehlenden Zeichenfolge 0-ende) an eine `'string'`-Funktion zum Absturz von UnoArduSim führen.
- 11) Der **CodeBereich** könnte die aktuelle Parse-Fehlerzeile im **falschen** Programmmodul hervorheben (wenn `'#include'` verwendet wurde).

- 12) Das Laden einer **'I / O' Geräte-Datei** mit einem Gerät, das (falsch) gegen 'Uno' Pin 13 gelenkt wurde, führte dazu, dass ein Programm im Popup-Fenster der Fehlermeldung hängen blieb.
- 13) UnoArduSim hatte dem Benutzer irrtümlich erlaubt, Nicht-Hex-Zeichen in die erweiterten TX-Pufferfenster für SPISLV und I2CSLV einzufügen.
- 14) Deklarationszeileninitialisierungen sind fehlgeschlagen, wenn der Wert auf der rechten Seite der **'return'** -Wert aus einer Objektelementfunktion war (wie in **'int angle = myservo1.read();'**).
- 15) **'static'** -**Mitglieder**- Variablen mit expliziten **'ClassName ::'** -**Präfixen** wurden nicht erkannt, wenn sie am Anfang einer Zeile (z. B. in einer Zuweisung zu einer base **'class'** -Variable) auftraten.
- 16) Das Aufrufen von **'delete'** auf einem Zeiger, der von **'new'** **erstellt wurde**, wurde nur erkannt, wenn die Funktionsklammernotation verwendet wurde, wie in **'delete (pptr)'**.
- 17) Die UnoArduSim-Implementierung von **'noTone()'** bestand fälschlicherweise darauf, dass ein Pin-Argument angegeben wurde.
- 18) Änderungen, die globale 'RAM' Bytes in einem Programm erhöhten, das **'String'** variables (über **Editieren / Ansehen** oder **Datei | Laden**) verwendete, konnte in diesem 'Uno' globalen Bereich aufgrund der Heap-Löschung der **'String'** Objekte zu einer **Beschädigung führen** zu dem alten Programm, während (fälschlicherweise) der Heap verwendet wird, der zu dem neuen Programm gehört. Dies könnte unter Umständen zu einem Programmabsturz führen. Obwohl ein zweites Laden oder Parsen das Problem gelöst hat, wurde dieser Bug endlich behoben.
- 19) Die Rückgabewerte für **'Wire.endTransmission()'** und **'Wire.requestFrom()'** waren beide bei 0 hängengeblieben - diese wurden nun behoben.

V1.7.2- Feb.2017

- 1) Interrupts auf Pin 2 wurden auch (versehentlich) durch Signalaktivität an Pin 3 ausgelöst (und umgekehrt).

V1.7.1- Feb.2017

- 1) Funktion **'delayMicroseconds()'** eine Verzögerung in **milli**-seconds herstellte. (1000 - mal zu groß).
- 2) Das explizite **Umwandeln** einer **'unsigned'** -**Variable** in einen längeren Ganzzahltyp ergab ein falsches (**'signed'**) Ergebnis.
- 3) Hex-Literalen, die größer als 0x7FFF sind, sind jetzt **'long'** per Definition, und so erzeugen sie nun **'long'** resultierende arithmetische Ausdrücke, an denen sie beteiligt sind.
- 4) Ein Fehler, der versehentlich durch V1.7.0 eingeführt wurde, verhinderte die alternative Umwandlung von numerischen Literalen in C++. (Beispiel: **'(long) 1000 * 3000'** wurde nicht akzeptiert).
- 5) **'Serial'** nimmt seine vielen Bytes in 'Uno' RAM nicht mehr auf, wenn es vom Anwenderprogramm nie benötigt wird.
- 6) Vom Benutzer deklarierte globale Variablen nehmen in 'Uno' RAM keinen Platz mehr ein, wenn sie niemals tatsächlich verwendet werden.
- 7) Einzelne Variablen, die als **'const'** „**'enum'**“ mitglieder deklariert sind, und Zeiger auf ZeichenfolgenLiteralen nehmen in 'Uno' RAM keinen Platz mehr ein (um der Arduino-Kompilierung zuzustimmen),
- 8) RAM-Byte, die für **'#include'** werden, schließen eingebaute Bibliotheken jetzt eng mit den Ergebnissen der bedingten Kompilierung von Arduino zusammen.
- 9) Mit **'new'** auf einem Zeiger war die tatsächliche Deklarationszeile fehlgeschlagen (nur eine spätere **'new'** Zuweisung zum Zeiger funktionierte).

10) Es wurde ein Fehler behoben, bei dem eine "ausstehende" Anzeige eines SD-Festplattenverzeichnisses zu einem Programmabsturz führen konnte.

V1.7.0- Dez.2016

0) Eine Reihe von Problemen mit der Behandlung von Benutzer-Interrupts wurde behoben:

a) Unterbricht 0 und 1 Kanten, die während einer Arduino-Funktion aufgetreten sind, die während des Wartens blockiert (wie `'pulseIn()'`, `'shiftIn()'`, `'spiTransfer()'`, `'flush()'` und `'write()'`) hatte einen Fehler im Ausführungsablauf bei der Interrupt-Rückgabe verursacht

b) Mehrere Kopien der lokalen Variablen irgendeiner unterbrochenen Funktion sind im **VariablenBereich** **erschienen** (eine Kopie pro Interrupt-Rückgabe) und dies wurde in V1.6.3 behoben, aber die anderen Interrupt-Probleme blieben bestehen.

c) Funktion `'delayMicroseconds()'` Es wurde keine Verzögerung erzeugt, wenn es innerhalb einer Benutzerunterbrechungsroutine aufgerufen wurde.

d) Anrufe zu blockieren Funktionen wie `'pulseIn()'` aus dem **Inneren** einer Interrupt - Routine hatte nicht gearbeitet.

1) Ein in V1.6.3 eingeführter Fehler verursachte den Verlust der Wertaktualisierung im **VariablenBereich** während der Ausführung, wenn sich die Werte tatsächlich änderten (dies geschah erst nach zwei oder mehr Benutzeraktionen von **Halt** oder Menü **VarAktualisieren**). Wenn ein Ausführen-Dort-Vorgang nach **Erlauben der Reduzierung** ausgelöst wurde, wurde der **VariablenBereich** gelegentlich nicht neu gezeichnet (alte Werte und lokale Variablen wurden dort möglicherweise bis zum nächsten Schritt angezeigt).

2) Der **CodeBereich** hervorheben Verhalten des **Schritt Über** Befehl könnte erscheinen in `'if()'` irreführend – `'sonst'` Ketten - das wurde jetzt behoben (obwohl die tatsächliche Schritt-Funktionalität korrekt war).

3) Funktion `'pulseIn()'` hatte das Zeitlimit in Millisekunden statt in Mikrosekunden falsch gesetzt - es war auch falsch das Zeitlimit erneut zu starten, wenn die Übergänge zu inaktiven und aktiven Ebenen zuerst gesehen wurden.

4) Die Verwendung von HEX-Literalen zwischen `0x8000` und `0xFFFF` in Zuweisungen oder Arithmetik mit `'long'` Integer-Variablen ergab falsche Ergebnisse aufgrund der nicht geprüften **Zeichenerweiterung**.

5) Das Übergeben oder Zurückgeben an einen `'float'` von einem beliebigen `'unsigned'` Integer-Typ mit einem Wert von MSB = 1 ergab falsche Ergebnisse aufgrund einer fehlerhaften `'signed'` Interpretation.

6) Alle `'bit _()'` -Funktionen akzeptieren nun auch Operationen an `'long'` -großen Variablen, und UnoArduSim prüft auf ungültige Bitpositionen (die außerhalb der Variablengröße liegen würden).

7) Eine ungültige Eingabe in die Impuls (Breite) -Bearbeitungsbox auf einem 'PULSER' -Gerät verursachte eine Beschädigung des 'Period' -Werts (bis durch den nächsten Benutzer **'Period'** -Bearbeitungseintrag festgelegt).

8) Das Löschen eines 'PULSER' oder 'FUNCGEN' -Geräts über das Menü Configure löschte nicht sein periodisches Signal von dem Pin, den es angesteuert hatte (ein Zurücksetzen ist nicht mehr erforderlich).

9) Die Möglichkeit, ein 1-D `'char'` -Array mit einem ZeichenfolgenLiteralen in Anführungszeichen zu initialisieren, fehlte (z. B. `'char strg[] = "Hallo";'`).

- 10) Die Hex-Anzeige in den erweiterten 'SERIAL' - oder 'SFTSER' Monitor-Fenstern zeigte das falsche höchstwertige Zeichen für Bytewerte größer als 127.
- 11) Die Wellenformen Fenster spiegelten keine programmgesteuerten Benutzeränderungen **wider**, die von 'analogWrite()' vorgenommen wurden, wenn der neue Wert entweder 0% oder 100% Einschaltdauer war.
- 12) Die Implementierung von 'Serial.end()' wurde jetzt behoben.
- 13) Eine Datei **myUnoPrefs.txt** mit mehr als 3 Wörtern in einer Zeile (oder Leerzeichen im 'I / O' Geräte-Dateinamen) könnte aufgrund eines fehlerhaften internen Zeigers einen Absturz verursachen.
- 14) Die letzte Zeile einer 'I / O' Geräte- Datei wurde nicht akzeptiert, wenn sie nicht **mit einem Zeilenvorschub beendet wurde** .
- 15) Das Hinzufügen von mehr als vier analogen Slidern verursachte einen stillen Fehler, der LED 'I / O' Geräte-Zeiger überschrieb
- 16) Beginnend mit V1.6.0 waren die analogen Wellenformproben für die **erste Hälfte** jeder **Dreieckswellenform** alle **null** (aufgrund eines Fehlers in der Wellenformtabellenberechnung).
- 17) Wenn Sie einen wiederholten Ausführen-Dort auf einen Haltepunkt ausführen, müssen Sie nicht mehr mehrere Klicks pro **Schritt**.
- 18) Das Übergeben von Adressausdrücken an einen Funktionsarrayparameter wurde vom Parser nicht akzeptiert.
- 19) Rekursive Funktionen, die Ausdrücke zurückgegeben haben, die Pointer- oder Array-De-Referenzen enthielten, ergaben falsche Ergebnisse, da die Flags "ready" für diese Komponentenausdrücke nicht zurückgesetzt wurden.
- 20) Der Aufruf von 'class' Mitglieder-functions über **eine Objektzeigervariable oder einen Zeigerausdruck** funktionierte nicht.
- 21) Benutzerfunktionen, die nur Objekte by-value zurückgaben, haben ihren Wert bei ihrem allerersten Funktionsaufruf zurückgegeben, **wenn** sie ein namenlos konstruiertes Objekt **zurückgaben** (wie 'String ("Hund") ' - bei nachfolgenden Aufrufen wurde die Rückgabe aufgrund eines "ready" übersprungen " Flagge.
- 22) Es gab keine Schutzmaßnahmen, um den Befehl **Windows** zu verhindern 'Serial' **Monitor**, um ein neues 'Serial' -Gerät **hinzuzufügen** , wenn tatsächlich kein Platz dafür war.
- 23) Wenn das Hinzufügen eines Fixed-Pin-Geräts (wie 'SPISLV') eine Popup-Meldung über einen Stiftkonflikt verursachte, konnte das **LaborBankBereich**- Redraw ein doppeltes "Ghost" -Gerät anzeigen, das das rechte 'I / O' -Gerät überlagert (bis zum nächsten Neuzeichnen). .
- 24) Einige Probleme mit unzuverlässigen 'PIEZO' Sounds für nicht-periodische Pin-Signale wurden behoben.
- 25) 'PROGMEM' Variablen müssen jetzt explizit als 'const' **deklariert werden** , um Arduino zuzustimmen.
- 26) "No heap space" wurde fälschlicherweise als Ausführungsfehler gekennzeichnet, wenn ein 'SD.open()' die angegebene Datei nicht finden konnte oder eine 'openNextFile()' die letzte Datei im Verzeichnis erreichte.

27) Ein Parser-Fehler hat eine schließende **geschweifte** GeschweifteKlammer '}' nicht ordnungsgemäß akzeptiert .

28) Ein Fehler beim Entfernen von **VariablenBereich** bei der Rückgabe von Mitglieder-Objekt-Konstruktoren wurde behoben (der Fehler wurde nur für Objekte angewendet, die selbst andere Objekte als Mitglieder enthalten).

V1.6.3- September 2016

1) Die lokalen Variablen einer unterbrochenen Funktion wurden nicht aus dem **VariablenBereich** bei der Eingabe der Interrupt-Funktion entfernt, was dazu führte, dass mehrere Kopien bei der Rückkehr der Interrupt-Funktion (und einem möglichen Ausführungsfehler oder einem Absturz) auftauchten.

2) Die Wellenform Fenster zeigten keine programmatischen Änderungen in '**analogWrite()**' zu einem neuen Arbeitszyklus von entweder 0% oder 100%.

3) Die Hex-Anzeige im erweiterten 'SERIAL' - oder 'SFTSER' Monitor-Fenster zeigte das falsche MSB-Zeichen für Bytewerte größer als 127.

V1.6.2- September 2016

1) Funktionsaufrufe, die mit der falschen Anzahl oder dem falschen Argumenttyp ausgeführt wurden, haben keine entsprechende Parse-Fehlernachricht generiert (nur die generische Meldung "Keine gültige Kennung" wurde angezeigt).

2) Die **Zurücksetzen-Taste** der Tool-Bar **funktioniert jetzt identisch mit der 'Uno' Leiterplatte Zurücksetzen-Taste**.

3) Parse-Fehler Text schneidet nicht mehr nach 16 Zeichen ab, ohne eine Ellipse zu zeigen.

V1.6.1- Aug. 2016

1) In V1.6 verursachte eine 'Uno' Leiterplatte-Version in der Datei **myArduPrefs.txt** , die sich vom Standardwert der Version 2 unterschied, beim Start eine Ausnahme (aufgrund eines nicht initialisierten Pin 13-Ereignisses).

2) Wenn Sie den Wert einer Variablen durch Doppelklicken im **VariablenBereich** ändern, kann dies zu fehlerhaften Fehlernachrichten "**keine Speicherzuweisung**" führen (für Programme mit einer benutzerdefinierten - '**class**').

3) '**SoftwareSerial**' erlaubte keinen Zugriff auf '**write (char * ptr)**' und '**write (byte * ptr, int size)**' Funktionen wegen einer fehlerhaften Funktionsüberlast-Erkennung.

4) Problem behoben, bei dem die entsprechende ".cpp" -Datei für einen isolierten .h "library-type" '**#include**' automatisch eingefügt wurde .

V1.6 - Juni 2016

1) In V1.5 war das automatische Einrücken auf der "Return" -Taste in **Editieren / Ansehen** (bei Eingabe einer neuen Zeile) verloren gegangen.

2) Die Erkennung von Pin-Konflikten mit angeschlossenen stark stromführenden externen 'I / O' -Geräten wurde nun auf '**Serial**' Pin 1, auf SPI-Pins SS, MOSI und SCK, auf I2C-Pins SCL und SDA hinzugefügt (alle beim zugehörigen '**begin()**' wird aufgerufen) und auf jedem angegebenen '**SoftwareSerial**' TX-Pin.

V1.5.1 - Juni 2016

1) In V1.5 wurden die neuen, an das Thema anpassbaren Syntax-Highlight-Farben bei jedem Öffnen von **Editieren / Ansehen** nicht richtig zurückgesetzt und waren daher (mit einem weißen Hintergrund-Thema) nur jedes zweite Mal korrekt.

2) Interrupt '**RISING**' und '**FALLING**' Empfindlichkeiten waren der tatsächlichen Polarität der **Triggerflanke** entgegengesetzt .

V1.5 - Mai 2016

1) Ein in V1.4.1 eingeführter Fehler verhinderte die Weitergabe von ZeichenfolgenLiteralen an Mitglieder-Funktionen, die ein '**String**' -Objekt erwarteten , wie in '**mystring1.startsWith ("Hey")**' .

2) Ein Fehler in der ursprünglichen **SD** Die Implementierung von UnoArduSim ermöglichte nur den **SD**-Zugriff über Aufrufe von '**read()**' und '**write()**' (Zugriff über '**Stream**' -Funktionen wurde verhindert).

3) '**R = 1K**' Schiebeschalter wurden nicht korrekt neu gezeichnet, wenn der Schieberegler bewegt wurde.

4) "**Stornieren**" im Bestätigen-Speicher Datei-Dialog sollte den Application-Exit verhindert haben.

5) Ein fehlendes Abschlusszitat oder das Schließen von '>' -Klammer in einer Benutzerdatei '**#include**' würde zu einem Absturz führen.

6) Es wurde ein Fehler bei der Syntaxhervorhebung von '**String**' und user '**class**' oder '**struct**' und der erweiterten Hervorhebung für Konstruktorfunktionsaufrufe **behooben** .

7) Kleinere Probleme in **Editieren / Ansehen** mit Textänderungen / Hervorhebungen und die Schaltfläche Rückgängig wurden behoben.

V1.4.3 - Apr. 2016

1) Verwenden von **Konfigurieren | 'I / O' Geräte** zum Hinzufügen neuer Geräte und zum anschließenden Entfernen eines dieser neu hinzugefügten Geräte können beim Zurücksetzen einen Absturz verursachen oder ein anderes Gerät nicht mehr funktionieren.

2) Das Ändern einer '**String**' -**Variablen** durch Doppelklicken im **VariablenBereich** ist fehlgeschlagen (der neue '**String**' wurde falsch gelesen).

3) Pin-Änderungen an den '**FUNCGEN**' - und '**PULSER**' -Geräten wurden erst erkannt, als ein Zurücksetzen durchgeführt wurde.

V1.4.2 - März 2016

1) V1.4.1 hatte einen unglücklichen Parse Fehler eingeführt , die Aufgaben , die in irgendeiner '**class**' Objekte verhindert (einschließlich '**String**').

2) Ein unvollständiger Bugfix in V1.4.1 führte dazu, dass '**unsigned**' -Werte vom Typ '**char**' als ASCII-Zeichen und nicht als Integer-Werte **ausgegeben wurden** .

3) Komplexe Aufrufargumente für Mitgliederausdrucksfunktionen wurden nicht immer als gültige Funktionsparameterübereinstimmungen erkannt.

4) Alle ganze Zha-Literale und -Ausdrücke waren zu großzügig bemessen (zu '**long**') und daher spiegelte die Ausführung nicht die tatsächlichen Überläufe (nach negativ) wider , die bei Arduino bei Add / Multiply-Vorgängen mit '**int**' sized-Werten auftreten können.

5) Ausdrücke mit einer Mischung aus '**signed**' und '**unsigned**' Integer-Typen wurden nicht immer korrekt behandelt (der '**signed**' -Wert würde **fälschlicherweise** als '**unsigned**' **angezeigt**).

6) In Pinkonfliktfällen können "value =" -Fehlermeldungen auch nach einem Zurücksetzen von einem früheren Konflikt, den der Benutzer bereits gelöscht hatte, veraltete Pinwerte anzeigen.

V1.4.1 - Jan. 2016

1) Aufrufe von '**print (char)**' **werden** jetzt korrekt als ASCII-Zeichen (anstelle von numerischen Werten) gedruckt.

2) Die Interrupt-Antwort ist jetzt standardmäßig aktiviert, wenn '**attachInterrupt()**' aufgerufen wird, so dass es keine Notwendigkeit mehr in Ihrem '**setup()**' gibt. Aufruf der Freischaltfunktion '**interrupts()**' .

3) Mehrere '**#include**' Instanzen von Benutzerdateien aus einer Datei werden nun korrekt behandelt.

V1.4 - Dez. 2015

1) Ein **langjähriger** Fehler wurde fälschlicherweise durch eine **Division durch Null gekennzeichnet** , wenn er durch einen Bruchwert kleiner als Eins dividiert wird.

2) Fixed '**SoftwareSerial**' (was versehentlich durch eine hinzugefügte '**class**' - Mitglieder-Validierungsprüfung in V1.3-Releases unterbrochen wurde).

3) End-of-Line-Funktionsaufrufe mit einem fehlenden Semikolon wurden nicht abgefangen und veranlassten das Parse dazu, die nächste Zeile zu überspringen.

4) Eine **falsch** formatierte '**I / O**' **Geräte-** Textdatei gab eine falsche Fehlermeldung aus.

5) Parse Error Hervorhebung der falschen (benachbarten) Zeile in mehrzeiligen Ausdrücken und Anweisungen wurde behoben

6) Logisches Testen von Zeigern mit dem '**not**' (**!**) Operator wurde invertiert.

V1.3 - Okt. 2015

1) Unsachgemäße interne Behandlung von Scratchpad-Variablen verursacht gelegentlich "**maximale Scratchpad Schachtelungstiefe überschritten**" Parse-Fehler.

2) Klammern in einfachen Anführungszeichen , Geschweifte Klammern. Semikolons, Klammern innerhalb von Zeichenfolgen in Anführungszeichen und Zeichen mit umgekehrter Schrägstrich - Präfix wurden falsch behandelt.

3) Ein Array mit einer leeren Dimension und keiner Initialisierungsliste führte zu einem RESET-Absturz, und Arrays mit nur einem einzelnen Element waren nicht unzulässig (und verursachten ihre fehlerhafte Interpretation als ungültiger initialisierter Zeiger).

4) Parse-Fehler würden manchmal manchmal die falsche (benachbarte) Zeile markieren.

5) Übergeben eines Zeigers an ein non '**const**' zu einer Funktion, die einen Zeiger auf eine '**const**' akzeptiert, wurde nicht **erlaubt** (statt umgekehrt).

6) Initialisiererausdrücke erben '**PROGMEM**' Qualifikationsmerkmale von der zu initialisierenden Variablen

- 7) '**PROGMEM**' deklarierte Variablen hatten ihre Byte-Größe **zweimal** falsch gegen ihre 'Flash' Speicherzuweisung während der Parse gezählt.
- 8) Eingabe in die 'Send' Eingabefelder eines 'I2CSLV' würde manchmal einen Absturz aufgrund von '**sscanf**' bug verursachen.
- 9) Das Laden eines neuen Programms mit einer neuen 'I / O' **Geräte**- Datei in seinem Verzeichnis kann zu irrelevanten Stiftkonflikten mit **alten** PIN-Anweisungen führen.
- 10) Im die Fenster zum Sehen 'Serial Monitor' TX und RX, die Behandlung von Zeichen mit umgekehrter Schrägstrich - Präfix wurde irrtümlich angewendet auf eingehende Zeichen, und nicht auf ausgehende Zeichen wurden .
- 11) '**while()**' und '**for()**' Schleifen mit vollständig leeren Körpern, wie '**while (true);**' oder '**for (int k = 1; k <= 100; k ++);**' bestanden die Parse (mit einem Warnmeldung), aber zur Ausführungszeit fehlgeschlagen.

V1.2 - Juni 2015

- 1) Die einfachsten Benutzerfunktionen, die Aufrufe an '**digitalRead()**' oder an '**analogRead()**' oder '**bit()**' vorgenommen haben, könnten ihre (allererste) deklarierte lokale Variable (falls vorhanden) aufgrund ungenügend zugewiesener Function Scratchpad beschädigt haben Leerzeichen (wenn nur zwei Scratchpad-Bytes am Anfang des Funktionsstacks zugewiesen wurden). Ein beliebiger numerischer Ausdruck innerhalb einer Funktion reicht aus, um eine 4-Byte-Notizblockzuteilung zu verursachen, und vermeidet so dieses Problem. Dieser unglückliche Fehler gibt es seit der ursprünglichen Version V1.0.
- 2) Funktionen, die '**void**' mit einer frühen expliziten '**return**' - und nicht 'void'-Funktionen mit mehr als einer '**return**' -Anweisung enthalten , würden bei der **schließenden GeschweifteKlammer** (falls sie erreicht wurde) einen Ausführungsdurchlauf sehen .
- 3) Alle '**return**' -Anweisungen in '**if()**' -Kontexten , in denen **keine** geschweiften GeschweifteKlammern vorhanden waren, führten zu einem fehlerhaften Ziel für die Rückkehr zum Aufrufer.
- 4) 'PULSER' und 'FUNCEN' Impulsbreiten oder Perioden mit dem Wert 0 können einen Absturz verursachen (0 ist jetzt nicht erlaubt).
- 5) Wo es keine GeschweifteKlammern gab, '**else**' Fortsetzungen nach einem '**if()**' funktionierten nicht, wenn sie einem '**break**' , '**continue**' oder '**return**' folgten .
- 6) Wenn **mehrere** '**enum**' Benutzer-Deklarationen gemacht wurden , generierten Konstanten, die in allen bis auf das erste '**enum**' definiert waren , fehlerhafte " '**enum**' mismatch" Parse-Fehler (dieser Fehler wurde in V1.1 eingeführt).
- 7) Ein Null-Bezeichner für den letzten Parameter eines Funktionsprototypen verursachte einen Parse-Fehler.
- 8) **Ausführen-Dort**- Haltepunkts, die in komplexen Zeilen gesetzt wurden, wurden nicht immer korrekt behandelt (und konnten daher auch nicht gefunden werden).
- 9) '**HardwareSerial**' und '**SoftwareSerial**' verwendete einen privaten TX-Pending-Puffer, der beim Zurücksetzen nicht bereinigt wurde (so dass übrig gebliebene Zeichen des letzten Mals angezeigt werden konnten).
- 10) Die Parse konnte nicht auf illegales Bit-Flipping von '**float**' prüfen , und die Zeigerarithmetik wurde mit unzulässigen Operatoren versucht.

V1.1 - März 2015

- 1) Array-Indizes, die **'byte'** - oder **'char'** -Size- Variablen waren, verursachten falsche Array-Offsets (wenn eine angrenzende Variable ein Nicht-0-High-Byte enthielt).
- 2) Das logische Testen von Zeigern testete den angezeigten Wert für Nicht-Null und nicht den Zeigerwert selbst.
- 3) Alle **'return'** -Anweisungen , die in **'for()'** oder **'while()'** Schleifen eingebettet sind, wurden falsch behandelt .
- 4) Aggregationsinitialisierungslisten für Arrays von Objekten oder Objekte, die andere Objekte / Arrays oder vollständig leere Initialisierungslisten enthalten, wurden nicht richtig behandelt.
- 5) Zugriff auf **'enum'** Mitglieder-Werte mit einem **" enumname"** Präfix wurde nicht unterstützt.
- 6) Die Deklarationszeileninitialisierung eines **'char[]'** -Arrays mit einem String-Literal in Anführungszeichen funktionierte nicht.
- 7) Ein Array, das ohne vorherige Initialisierung an eine Funktion übergeben wurde, wurde fälschlicherweise mit einem "used but not initialized" -Fehler gekennzeichnet.
- 8) Zeigerausdrücke mit Array-Namen wurden falsch behandelt.
- 9) Funktionsparameter, die als **'const'** deklariert wurden, wurden nicht akzeptiert.
- 10) Das Pin AnalogWellenformFenster zeigte keine PWM-Signale an (**'servo.write()'** und **'analogWrite()'**).
- 11) Mitgliederfunktionen, auf die über einen Objektzeiger zugegriffen wurde, ergaben fehlerhafte Mitgliederzugriffe.
- 12) Wellenformen wurden nicht aktualisiert, wenn ein **Ausführen-Dort-** Haltepunkt erreicht wurde.
- 13) Die Registerzuordnungsmodellierung kann fehlschlagen, wenn ein Funktionsparameter direkt als Argument für einen anderen Funktionsaufruf verwendet wurde

V1.0.2 - Aug. 2014

Die Reihenfolge der A0-A5-Pins am Umfang der 'Uno' -Platine wurde korrigiert .

V1.0.1 - Juni 2014

Es wurde ein Fehler behoben, durch den Bearbeitungspasten abgeschnitten wurden, die länger waren als das Dreifache der Anzahl der Bytes im ursprünglichen Programm.

V1.0 - erste Veröffentlichung Mai 2014

Änderungen / Verbesserungen

V2.0.1- Jan. 2018

- 1) Undokumentierte Arduino-Funktionen '**exp()**' und '**log()**' wurden jetzt hinzugefügt.
- 2) 'SERVO'-Geräte können jetzt kontinuierlich gedreht werden (so steuert die Pulsbreite die Geschwindigkeit anstelle des Winkels).
- 3) In **Editieren / Ansehen** wird eine schließende Klammer '**}**' jetzt automatisch hinzugefügt, wenn Sie eine öffnende Klammer '**{**' eingeben.
- 4) Wenn Sie zum Beenden auf die Titelleiste '**X**' des **Editieren / Ansehen**-Fensters klicken, haben Sie jetzt die Möglichkeit abzubrechen, wenn Sie die angezeigte Programmdatei geändert, aber nicht gespeichert haben.

V2.0 Dez. 2017

- 1) Die Implementierung wurde in QtCreator portiert, so dass die GUI einige kleinere visuelle Unterschiede aufweist, aber abgesehen von einigen Verbesserungen keine funktionalen Unterschiede:
 - a) Die Benachrichtigung über die Statuszeile am unteren Rand des Hauptfensters und im Dialogfeld "**Editieren / Ansehen**" wurde verbessert und eine hervorgehobene Farbcodierung hinzugefügt.
 - b) Der vertikale Bereich, der zwischen dem **CodeBereich** und dem **VariablenBereich** zugewiesen ist, ist nun durch eine ziehbare (aber nicht sichtbare) Trennleiste an ihrem gemeinsamen Rand einstellbar.
 - c) 'I / O' Geräte-eingabefeld-Werte werden jetzt nur dann validiert, wenn der Benutzer den Mauszeiger aus dem Gerät bewegt hat - dies vermeidet umständliche automatische Änderungen, um legale Werte zu erzwingen, während der Benutzer tippt.
- 2) UnoArduSim unterstützt jetzt mehrere Sprachen über **Konfigurieren | Präferenzen**. Neben der Sprache für das Benutzergebietsschema kann immer Englisch ausgewählt werden (sofern eine benutzerdefinierte *.qm-Übersetzungsdatei für diese Sprache im Ordner UnoArduSim '**translations**' vorhanden ist).
- 3) Sound wurde nun modifiziert, um die Qt-Audio-API zu verwenden - dies erforderte unter bestimmten Umständen eine Stummschaltung, um störende Soundunterbrechungen und Klicks während der durch normale Mausklicks verursachten längeren OS-Windowing-Verzögerungen zu vermeiden. Weitere Informationen finden Sie im Abschnitt Modellierung-Sounds Detail dazu.
- 3) Aus praktischen Gründen werden Leerzeichen jetzt in den Eingabefeldern als Null für die Anzahl der Geräte in **Konfigurieren | 'I / O' Geräte** (Sie können jetzt die Leertaste verwenden, um Geräte zu entfernen).
- 5) Das unskalierte (U) Qualifikationsmerkmal ist jetzt optional für 'PULSER', 'FUNCGEN' und '1SHOT'-Geräte (dies ist der angenommene Standardwert).
- 6) UnoArduSim ermöglicht jetzt (zusätzlich zu literalen numerischen Werten) '**const**' - ganzzahlige Variablen und '**enum**' -**Mitglieder**, die als Dimensionen in Array-Deklarationen verwendet werden, solange diese Quellen explizit mit einem ganzzahligen numerischen Literal initialisiert werden Konstanten.
- 7) Nachdem die Maus zuerst eintritt, wird eine Schiebevorrichtung jetzt erzeugt Kontakt (für 1 Millisekunde) Prellen auf jeder Betätigung der **Leertaste** Taste (aber **nicht** für Mausklicks, noch für anderen Tastendruck).
- 8) Weitere Klicks von **Finden | Setze SuchText** wählt nun den nächsten aus Wort aus dem Text der hervorgehobenen Zeile im aktiven Bereich (der **CodeBereich** oder der **VariablenBereich**).
- 9) Der '**goto**' Arduino-Befehl wird jetzt in UnoArduSim als "nicht unterstützt" gekennzeichnet.
- 10) Funktionen '**max()**' und '**pow()**' wurden jetzt in die Bequemlichkeit einbezogen Liste von Einbauten innerhalb des **Editieren / Ansehen**- Dialogs.

V1.7.2- Feb. 2017

1) Die Farbauswahl Blau (B) wurde für LED-Geräte hinzugefügt.

V1.7.1- Feb. 2017

1) Suffixe '**L**' und / oder '**U**' werden nun am Ende numerischer **Literalkonstanten** akzeptiert (um sie als '**long**' und / oder '**unsigned**' zu definieren)) und (**0b** oder **0B** vorangestellt) binäre Konstanten werden nun ebenfalls akzeptiert. Jede numerische Ganzzahl, die **mit einem** '0' beginnt, wird jetzt als **Oktalwert betrachtet** . (mit Arduino zuzustimmen).

2) Bei der Ausführung in einer Tight-Schleife, von der es kein Ausbruch gibt (z. B. '**while (x); x ++;**' wobei x immer wahr ist), wird durch Klicken auf **Halt** ein zweites Mal die Programmausführung angehalten (und auf dieser fehlerhaften Programmzeile) .

V1.7.0- Dez.2016

1) Eine neue **Toolbar**- Funktion wurde hinzugefügt, die freie RAM-Bytes während der Programmausführung anzeigt (unter Berücksichtigung der gesamten Bytes, die von globalen Variablen, Heap-Zuordnungen und lokalen Stack-Variablen verwendet werden).

2) Benutzerunterbrechungsfunktionen können jetzt auch selbst blockierende Arduino-Funktionen wie '**pulseIn()**' aufrufen (dies sollte jedoch nur mit Vorsicht verwendet werden, da die Interruptfunktion erst zurückkehrt, wenn die Blockierfunktion abgeschlossen ist).

3) Benutzer-Interrupts sind während blockierter Stream-Lese-Operationen nicht mehr deaktiviert, daher entspricht das Verhalten jetzt dem tatsächlichen Arduino-Stream-Lese-Vorgang.

4) Sie können nun **Schritt -In** und **Aus** dem Blockieren von Arduino-Funktionen, die unterbrochen werden können (wie '**delay()**' und '**pulseIn()**'), und Status-Bar-Nachrichten wurden erweitert, um anzuzeigen , wenn Sie einen Interrupt-Haltepunkt innerhalb einer solchen Funktion (oder wenn Sie klicken-Halt, wenn die Ausführung gerade in einer solchen Funktion ist) getroffen haben.

5) Ein neuer **Ausführen-Bis**- Befehl (und **Tool-Bar**- Element) wurde hinzugefügt - klicken Sie einfach auf eine **Variablen** Variable (es kann einfach sein, ein Aggregat-Array oder Objekt, oder ein Array-Element oder Objekt-Mitglied), um es hervorzuheben, dann wird **Ausführen-Bis** ausgeführt - die Ausführung wird beim nächsten **Schreibzugriff** innerhalb dieser Aggregatvariablen oder an diesem einzelnen Ort eingefroren .

6) Wenn die Ausführung nach einem **Schritt** gefriert, **Ausführen-Dort**, **Ausführen-Bis**, oder **Ausführen** -dann- **Halt** Aktion, Fenster , um die Variablen nun hebt die Variable, die an die Adresse Lage entspricht (en) , die (falls vorhanden) durch die sehr zuletzt geändert wurden Anweisung Während dieser Ausführung - wenn diese Position derzeit in einem nicht expandierten Array oder Objekt verborgen ist, bewirkt das Anklicken, um sie zu erweitern, dass das zuletzt geänderte Element oder Element zuletzt hervorgehoben wird.

7) Der Benutzer kann nun den Wert einer bestimmten variablen Bereichsvariablen / Mitglied / Element während der Ausführung überwachen - doppelklicken Sie auf diese Zeile im **VariablenBereich** , um das Fenster zum **Editieren / Verfolgen** von Variablenwerten zu öffnen , und führen Sie dann einen der folgenden **Ausführen** aus oder **Schritt** befehle - Der angezeigte Wert wird während der Ausführung gemäß denselben Regeln aktualisiert, die für Aktualisierungen im **VariablenBereich** gelten . Nach der Ausführung Einhalt zu gebieten, ist . Sie erlaubt einen neuen Wert eingeben und übernahm vor dem Fortsetzen der Ausführung (und den Wert **Revert** kann im Voraufbau **akzeptieren** , wenn Sie Ihren Kopf , bevor dann ändern).

8) Die Accelerator-Tasten F4-F10 wurden so eingestellt, dass sie den **Tool-Bar**- Befehlen des Execute-Menüs entsprechen (von links nach rechts).

9) Zusätzlich zu einem Doppelklick auf sie wird mit einem Rechtsklick auf 'SERIAL', 'SFTSER', 'SPISLV', 'I2CSLV' Geräte auch ein größeres TX / RX Byte / Zeichen Fenster (und auf 'SD_DRV) angezeigt', ein Datei-Überwachungsfenster).

10) Die TX-eingabefeld in 'SERIAL' oder 'SFTSER' ist während einer aktiven Zeichenübertragung nicht mehr deaktiviert (Sie können nun anhängen oder ersetzen, was dort vorhanden ist), aber ein Wagenrücklauf (oder 'Send' -Knopf in der associated 'SerialMonitor' child fenster) wird solange ignoriert, bis die Übertragung wieder in den Ruhezustand zurückkehrt (Zeichen werden nun kursiv dargestellt, wenn die Übertragung bereit ist, ot ist aktiv). Außerdem wird der Benutzer jetzt bei einem seriellen Stream **gewarnt** **'begin()'** wenn sie schon früher mit den angehängten Geräten begonnen hätten (jetzt in Bearbeitung), dann würde es keine Framing-Synchronisation geben, was zu Empfangsfehlern führen würde.

11) Die standardmäßig hinzugefügte **'loop()'** Verzögerung wurde von 250 Mikrosekunden auf eine Millisekunde erhöht, um nicht so weit hinter die Echtzeit zu fallen, wenn der Benutzer es versäumt, irgendwo **'delay()'** (explizit oder natürlich) einzubeziehen innerhalb **'loop()'** oder innerhalb einer Funktion.

12) Arrays und einfache Typen wurden nun der Unterstützung für die Heap-allocating **'new'** Anweisung **hinzugefügt**.

13) Umfangreichere Überprüfungen (und damit verbundene Fehlermeldungen) wurden für die Zugriffe außerhalb der Grenzen des Benutzerprogramms hinzugefügt (dh außerhalb von 'Uno' RAM oder außerhalb von 'Flash' für **'PROGMEM'** Zugriffe).

14) Zeigerwerte im **VariablenBereich** ähneln nun eher den tatsächlichen Arduino-Zeigerwerten.

15) Die Datei **myArduPrefs.txt** des Benutzers wird jetzt bei jeder gelesen **Datei | Laden**, nicht nur beim Starten von UnoArduSim.

16) Ein Parse-Fehler wird jetzt beim Versuch, **'attachInterrupt()'** markiert zu einer Benutzerfunktion, die nicht **'void'** **ist** zurück, oder die Funktionsparameter hat, oder die vor **'attachInterrupt()'** noch nicht deklariert wurde.

17) **'static'** Mitglieder-variables werden jetzt am oberen Rand des **VariablenBereichs** als globale **Variablen** angezeigt, anstatt innerhalb jeder Instanz eines (erweiterten) Objekts angezeigt zu werden.

18) Funktion **'availableForWrite()'** wurde der Implementierung von `Serial` hinzugefügt.

19) Alle speziellen **'PROGMEM'**, **'typedef'** like **'prog_char'** und **'prog_int16'** wurden jetzt entfernt (sie wurden in Arduino veraltet).

20) Verbesserte Fehlermeldungen für Parse-Fehler, die durch falsch geschriebene oder ungültige Deklarationstypen verursacht wurden.

21) Die maximal zulässige Programmgröße wurde erhöht.

V1.6.3- September 2016

1) Es wurde eine verbesserte **Syntaxanalyse**- Fehlermeldung **hinzugefügt**, wenn sich **'attachInterrupt()'** auf eine Interrupt-Funktion bezieht, die **zuvor** nicht **prototypisiert wurde**.

2)

3) Eine verbesserte Parse-Fehlermeldung für mehrdimensionale Array-Initialisierungslisten wurde hinzugefügt.

V1.6.2- September 2016

- 1) **Finden-Text-** Edit-Steuerelement zur **Symbolleiste** hinzugefügt , um die Suche nach Text zu rationalisieren (im **CodeBereich** und im **VariablenBereich**).
- 2) Die Schaltfläche Zurücksetzen der **Werkzengleiste** funktioniert jetzt identisch mit der Zurücksetzen-Taste 'Uno' Leiterplatte.

V1.6.1- August 2016

Eine Überprüfung hinzugefügt, um das doppelte Laden und Parsing von bereits früheren **'#include'** -Dateien zu vermeiden .

V1.6 - Juni 2016

- 1) Ein neues '1SHOT' (Generator Ein-Schuss) 'I / O' Geräte wurde hinzugefügt, das nach einer gewählten Verzögerung von einer Trigger-Signalfanke mit ausgewählter Polarität einen Impuls erzeugt.
- 2) Neue Funktion hinzugefügt, um 'I / O' Geräte-eingabefeld-Werte während der Ausführung leicht **skalierbar** zu machen, indem ein globaler 'I / O _____ S' Scaler-Schieberegler in der Haupt-**Werkzengleiste gezogen wird** (einfach einen Buchstaben eingeben 's' oder 'S' nach einem Wert zur Angabe der Skalierung).

V1.5.1 - Juni 2016

- 1) Unterstützung wurde jetzt für die EEPROM-Bibliotheksfunktionen **'update()'** , **'put()'** und **'get()'** hinzugefügt , und für Byte-Zugriff über Array-Notation, z **'EEPROM [k]'** .
- 2) **Erlauben Auto(-) Collapse** wurde dem Menü **VarAktualisieren** hinzugefügt, um eine explizite Kontrolle darüber zu ermöglichen, ob expandierte Arrays / Objekte automatisch kollabiert werden, wenn die Ausführung hinter Echtzeit zurückbleibt.
- 3) Die Charaktere eines **'string'** Variable kann jetzt auch zugegriffen werden über Array-Notation, z. B **'mystring [k]'** .

V1.5 - Mai 2016

- 1) **Ansehen / Editieren** verfügt jetzt über die Tastenkombination Strg-E und über eine neue Schaltfläche zum **Kompilieren** (Strg-R) sowie ein integriertes Feld zum Analysieren von Fehlern, um das Testen von Änderungen zu ermöglichen, ohne das Fenster schließen zu müssen.
- 2) **Ansehen / Editieren** unterstützt jetzt auch **Wiederholen** und hat eine neue Schaltfläche **Speichern** (Strg-S) (entspricht **Accept** plus einem späteren Hauptfenster **Speichern**) und gibt nun eine Auswahl der TAB-Größe (eine neue Einstellung, die mit gespeichert werden kann) **Konfigurieren | Präferenzen**).
- 3) Alle beschreibbaren Eingabefelder folgen jetzt den ausgewählten Windows-Betriebssystem-Designfarben, und zum Kontrast verwenden alle schreibgeschützten 'RECV' Eingabefelder weißen Text auf schwarzem Hintergrund. Die **Editieren / Ansehen-** Hintergrund- und Syntax-Highlight-Farben passen sich nun auch dem gewählten Thema an.
- 4) UnoArduSim erlaubt jetzt die Auswahl einer Schriftart - diese Auswahl und ihre Größe wurden in **Konfigurieren | Präferenzen** verschoben (können also in der Datei **myArduPrefs.txt** gespeichert werden).
- 5) Arduino vordefinierte binäre **Literalwerte** (wie **'B01011011'**) sind jetzt erlaubt.
- 6) Hex-zeichen, oktal-Zeichen, und 4-stellige Unicode-Anführungszeichen Folgen (alle mit umgekehrter Schrägstrich - Präfix) können jetzt als numerische Literalen verwendet werden.

7) Nach einem ersten Mausklick auf ein 'PUSH' Geräte-Push-Pad kann der Benutzer stattdessen einen Tastendruck (beliebige Taste) drücken, um die Druckknopfkontakte zu drücken.

8) **Editieren / Ansehen gibt** jetzt seinen temporären anfänglichen schreibgeschützten Zustand frei (und entfernt die Hervorhebung der ursprünglichen ausgewählten Zeile) nach einem kurzen visuellen Flash-Cue.

9) UnoArduSim prüft nun auf mehrere '**Stepper**' - und '**Servo**' -**Pin**- Konflikte, dh fehlerhafte Benutzerprogramme versuchen, an bereits an frühere '**Stepper**' - oder '**Servo**' -**Variablen** angehängten Pins anzuhängen .

10) Ein Parse-Fehler, der durch eine fehlende linke oder rechte Seite eines Operators verursacht wird (wobei ein LHS- oder RHS-Ausdruck oder eine LSE-Variable fehlt), erzeugt nun eine eindeutige Fehlermeldung.

11) Die nicht verwendete '**String**' class '**flags**' **Elementvariable** wurde entfernt, um Arduino V1.6.6 zuzustimmen. Ein '**String**' -Objekt belegt jetzt 6 Bytes (plus seine Zeichen-Heap-Zuweisung).

V1.4.2 - März 2016

1) Vorwärtsdefinierte Funktionen (dh solche ohne Prototypdeklaration vor ihrem ersten Aufruf) erzeugen nun nur Warnungen (keine Parse-Fehler), wenn der spätere Funktionsdefinitions-Rückgabebetyp dem von ihrer ersten Verwendung abgeleiteten Typ nicht entspricht.

2) Arrays mit einer Dimension von 1 werden nicht mehr abgelehnt (um den Standard-C ++ - Regeln zu entsprechen).

3) Eingabefelder sind nicht mehr schwarz auf weißem Hintergrund - sie übernehmen jetzt die Palette, die vom verwendeten Windows-Betriebssystem verwendet wird.

4) 'SERIAL', 'SFTSER', 'SPISLV' und 'I2CSLV' Gerät erweitert Monitorfenster (durch Doppelklick geöffnet) nehmen nun die Hintergrundfarbe ihres Eltern 'I / O' Geräte an.

V1.4 - Dez. 2015

1) '**Stepper.h**' library-Funktionalität und zugehörige 'I / O' -Geräte wurden hinzugefügt.

2) **Alle 'I / O' Geräteeinstellungen und -werte** (zusätzlich zu den ausgewählten Pins) werden jetzt auch als Teil der ausgewählten Benutzer 'I / O' Geräte Textdatei zum späteren Neuladen gespeichert.

3) **LED** 'I / O' Gerätefarbe kann nun über eine eingabefeld am Gerät entweder rot, gelb oder grün eingestellt werden.

4) Variablendeklarationsinitialisierer dürfen jetzt mehrere Zeilen umfassen.

5) Array-Indizes dürfen nun selbst Array-Elemente sein.

6) **Konfigurieren | Präferenzen** enthält nun eine Checkbox , die es erlaubt, '**and**' , '**or**' , '**not**' Schlüsselwörter anstelle des C-Standards **&&** , **||** zu verwenden und ! logische Operatoren.

7) "Zeigen ProgrammHerunterladen" wurde in verschoben **Konfigurieren | Präferenzen**

V1.3 - Okt 2015

- 1) Das **'PUSH'** -Gerät hat jetzt ein "push-like" -Kästchen mit der Aufschrift 'latch', um sie "rastend" (statt "momentan") zu machen, das heißt, sie werden in der geschlossenen Position einrasten (und die Farbe wechseln) wenn gedrückt, bis sie erneut gedrückt werden, um die Kontakte freizugeben.
- 2) Vollständige Fähigkeiten 'SPISLV' Geräte wurden mit Knotenauswahl hinzugefügt (**'MODE0'** , **'MODE1'** , **'MODE2'** oder **'MODE3'**). Ein Doppelklick öffnet ein TX / RX-Pufferfenster, in dem anstehende REPLY (TX) -Bytes definiert werden können und um vergangene empfangene (RX) Bytes anzuzeigen. Das einfache Schieberegisterr-Sklave Gerät der Vorgängerversion wurde umbenannt in ein 'SRSLV' -Gerät.
- 3) **Fettschrift kann nun für den CodeBereich** und den **VariablenBereich** (über das Menü Optionen) *ausgewählt werden, und die* Hervorhebung von Schlüsselwörtern und Operatoren kann fett hervorgehoben werden **kann jetzt in Editieren / Ansehen** ein- und ausgeschaltet werden.
- 4) UnoArduSim erlaubt nun **'bool'** als Synonym für **'boolean'** .
- 5) Aus Gründen der Klarheit bei der Fehlerberichterstattung dürfen Variablendeklarationen nicht länger mehrere Zeilen umfassen (außer für Arrays mit Initialisierungslisten).
- 6) Syntaxfärbung in **Editieren / Ansehen** wurde verbessert (dies wird bei größeren Programmen auffallen).
- 7) Ein optionaler 200-Mikrosekunden-Overhead (im Menü **Optionen**) wurde zu jedem Aufruf von **'loop ()'** hinzugefügt - um zu vermeiden, zu weit hinter die Echtzeit zu fallen, wenn das Benutzerprogramm keine hinzugefügt hat **'delay ()'** anywhere (siehe Zeitliche Koordinierung Diskussion unter **Modellierung**).

Version 1.2 Juni 2015

- 1) Die SD-Bibliothek ist nun vollständig implementiert und ein (kleines) 8 MByte SD Disk 'I / O' -Gerät ('SD_DRV') wurde hinzugefügt (und die Funktionalität wurde für alle SD-Programme von Arduino-Beispiel getestet).
- 2) Wie Arduino konvertiert UnoArduSim jetzt automatisch ein Funktionsargument in seine Adresse, wenn es eine Funktion aufruft, die erwartet, dass ein Zeiger übergeben wird.
- 3) Parse-error-Meldungen sind jetzt besser geeignet, wenn Semikolons fehlen und nach nicht erkannten Deklarationen.
- 4) Veraltete **Variablen Fensterlinien** -Hervorhebungen werden jetzt beim Funktionsaufruf / -rücklauf entfernt.

V1.1 - März 2015

- 1) Das Hauptfenster kann jetzt vergrößert oder verkleinert werden, um den **CodeBereich** und den **VariablenBereich** breiter zu machen (für größere Bildschirme).
- 2) Ein neues Menü **Finden** (mit Tool-Bar - Tasten) wurde hinzugefügt schneller Navigation im **CodeBereich** und **VariablenBereich** (mit Pfeil nach oben, Pfeil nach unten Bild hoch und Bild ab oder Textsuche) zu ermöglichen.
- 3) Das **Ansehen / Editieren** fenster erlaubt nun Strg-PgUp und Strg-PgDn Navigationssprünge (zur nächsten Leerzeile) und hat die **Finden/ Ersetzen** Funktionalität erweitert.

- 4) Ein neues Element wurde dem Menü **VarAktualisieren** hinzugefügt, um es dem Benutzer zu ermöglichen, einen Berechnungsspar-Ansatz unter schweren **Variablen**- Aktualisierungsbelastungen auszuwählen .
- 5) 'Uno' -Pins und die beigefügte LED geben nun alle Änderungen an 'I / O' -Geräten wieder, selbst wenn die Zeit eingefroren ist (dh selbst wenn die Ausführung angehalten wurde).
- 6) Andere Benutzerfunktionen können jetzt von innerhalb einer Benutzerunterbrechungsfunktion (in Übereinstimmung mit der Aktualisierung auf Arduino 1.06) aufgerufen werden.
- 7) Eine **größere Schriftart** kann nun aus dem Menü **Optionen ausgewählt werden** .

V1.0.1 - Juni 2014

Wellenformen Fenster bezeichnen analoge Pins jetzt als A0-A5 anstelle von 14-19.

V1.0 - erste Veröffentlichung Mai 2014