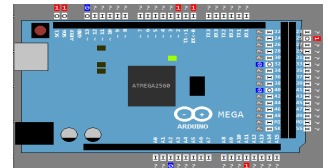


UnoArduSimV2.8.1 Volle Hilfe



Inhaltsverzeichnis

Überblick

CodeBereich, Präferenzen und Editieren/Ansehen

CodeBereich

Präferenzen

Editieren/Ansehen

VariablenBereich und Editieren/Verfolgen Variable fenster

LaborBankBereich

Das 'Uno' oder 'Mega'

'I/O' Geräte

'Serial' Monitor ('SERIAL')

Wechseln Seriennummer ('ALTSER')

SD-Laufwerk ('SD_DRV')

TFT Bildschirm ('TFT')

Konfigurierbares SPI Sklave ('SPISLV')

Zweileiter I2C Sklave ('I2CSLV')

Text LCD I2C ('LCDI2C')

Text LCD SPI ('LCDSPI')

Text LCD D4 ('LCD_D4')

Multiplexer LED I2C ('MUXI2C')

Multiplexer LED SPI ('MUXSPI')

Erweiterungsport SPI ('EXPSPi')

Erweiterungsport I2C ('EXPI2C')

'1-Wire' Sklave ('OWIISLV')

Schieberegister Sklave ('SRSLV')

Generator Ein Schuss ('1SHOT')

Programmierbares 'I/O' Gerät ('PROGIO')

Digitaler Impulsgeber ('PULSER')

Analoge Funktionsgenerator ('FUNCGEN')

Schrittmotor ('STEPR')

Gepulst Schrittmotor ('PSTEPR')

GleichstromMotor ('MOTOR')

ServoMotor ('SERVO')

Piezo Lautsprecher ('PIEZO')

Schiebewiderstand ('R=1K')

Druckknopf ('PUSH')

Farbige LED ('LED')

4-LED Row ('LED4')

7-Segment LED Ziffer ('7SEG')

Analoger Schieberegler

Pin Drahtbrücke ('JUMP')

Menüs

Datei:

Laden INO oder PDE Prog (Strg-L)

Editieren/Ansehen (Strg-E)

Speichern

Speichern Als

Nächster ('#include')

Bisherige

Ausgang

Finden:

Aufsteigen Anruf-Stapel

Absteigen Anruf-Stapel

Setze Suche Text (Strg + F)

inden Nächster Text

[Finden Vorheriger Text](#)

[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 Tragwerke/ Operatoren](#)
[Registerzuordnung](#)
[Fehler bei Nicht initialisiert](#)
[Hinzugefügt 'loop\(\)' Verzögern](#)
[Verschachtelte Interrupts zulassen](#)

[Konfigurieren:](#)

['I/O' Geräte](#)
[Präferenzen](#)

[VarAktualisieren:](#)

[Erlaube Auto \(-\) Zusammenziehen](#)
[Minimal](#)
[Markieren Änderungen](#)

[Fenster:](#)

['Serial' Monitor](#)
[Alles wiederherstellen](#)
[Pin Digital Wellenformen](#)
[Pin Analoge Wellenform](#)

[Hilfe:](#)

[Schnell Hilfe Datei](#)
[Volle Hilfe Datei](#)
[Fehler-Korrekturen](#)
[Änderung / Verbesserungen](#)
[Über](#)

['Uno' oder 'Mega' Leiterplatte und 'I/O' Geräte](#)

[Timing](#)

['I/O' Gerät Timing](#)

[Geräusche](#)

[Einschränkungen und nicht unterstützte Elemente](#)

[Enthaltenes Dateien](#)

[Dynamische Speicherzuordnungen und RAM](#)

['Flash' Speicherzuordnungen](#)

['String' Variablen](#)

[Arduino-Bibliotheken](#)

[Zeiger](#)

['class' und 'struct' Objekte](#)

[Imfang](#)

[Qualifikanten 'unsigned', 'const', 'volatile', 'static'](#)

[Kompilierer-Richtlinien](#)

[Arduino-sprachige Elemente](#)

[C / C ++ - Sprachelemente](#)

[Funktionsmodul Vorlagen](#)

[Echtzeit-Emulation](#)

[Versionshinweise](#)

[Fehler-Korrekturen](#)

[V2.8.1- Juni 2020](#)

[V2.8.0- Juni 2020](#)

[V2.7.0- März 2020](#)

[V2.6.0- Jan 2020](#)

[V2.5.0- Oktober 2019](#)

[V2.4 - Mai 2019](#)

[V2.3 - Dezember 2018](#)

[V2.2– Jun. 2018](#)

[V2.1.1– März 2018](#)

[V2.1– März 2018](#)

[V2.0.2 Feb. 2018](#)

[V2.0.1– Jan. 2018](#)

[V2.0– Dez. 2017](#)

[Änderungen / Verbesserungen](#)

[V2.8.0- Juni 2020](#)

[V2.7.0- März 2020](#)

[V2.6.0 Januar 2020](#)

[V2.5.0 Oktober 2019](#)

[V2.4 Mai 2019](#)

[V2.3 Dez. 2018](#)

[V2.2 Juni 2018](#)

[V2.1 März 2018](#)

[V2.0.1 Jan. 2018](#)

[V2.0 Sept. 2017](#)

Überblick

UnoArduSim ist eine Freeware **Echtzeit** (sehen für Timing **Beschränkungen**) Simulator-Tool, das ich für den Studenten und Arduino-Enthusiasten entwickelt habe. Es ist so konzipiert, dass Sie mit Arduino programme experimentieren und problemlos debuggen können **ohne die Notwendigkeit einer tatsächlichen Hardware**. Es richtet sich an die **Arduino 'Uno' oder 'Mega'** Mit leiterplatte und können Sie aus einer Reihe von virtuellen 'I/O' geräte auswählen und diese geräte konfigurieren und mit Ihrem virtuellen 'Uno' oder 'Mega' im verbinden **LaborBankBereich**. - Sie müssen sich keine Sorgen über Verdrahtungsfehler, unterbrochene / lose Verbindungen oder fehlerhafte geräte machen, die Ihre programm-Entwicklung und -Tests durcheinander bringen.

UnoArduSim bietet einfache Fehlermeldungen für alle aufgetretenen analysieren- oder ausführung-Fehler und ermöglicht das Debuggen mit **Zurücksetzen**, **Ausführen**, **Ausführen Dort**, **Ausführen Bis**, **Halt** und flexibel **Schritt** Operationen in der **CodeBereich**, mit einer gleichzeitigen Ansicht aller globalen und derzeit aktiven lokalen variablen, arrays und objekte im **VariablenBereich**. Ausführen-malige array-Grenzüberprüfung wird bereitgestellt und ein ATmega-RAM-Überlauf wird erkannt (und die programm-Täterzeile hervorgehoben!). Alle elektrischen steht konflikt mit, die an 'I/O' geräte angeschlossen sind, werden markiert und gemeldet, sobald sie auftreten.

Wenn eine INO oder PDE programm datei geöffnet wird, wird sie in die programm geladen **CodeBereich**. Der programm erhält dann eine Analysieren, um es in eine tokenisierte ausführbare Datei umzuwandeln, für die es dann bereit ist **simulierte ausführung** (Im Gegensatz zu Arduino.exe ist eine eigenständige ausführbare binären-Datei *nicht* Jeder analysieren - Fehler wird erkannt und markiert, indem die Zeile markiert wird, die bei analysieren fehlgeschlagen ist, und der Fehler auf dem gemeldet wird **Statusleiste** Ganz unten in der UnoArduSim-Anwendung fenster. Ein **Editieren/Ansehen** fenster kann geöffnet werden, damit Sie eine syntaktisch hervorgehobene Version Ihres Benutzers programm anzeigen und bearbeiten können. Fehler während des simulierten ausführung (z. B. ein falsch zugeordneter baudrate) werden in der Statusleiste und über ein Popup-Meldungsfeld gemeldet.

UnoArduSim V2.8 ist eine im Wesentlichen vollständige Implementierung des **Arduino Programming Language V1.8.8 wie am dokumentiert arduino.cc**. Sprachreferenz-Webseite und mit Ergänzungen, wie in den Versionshinweisen der Herunterladen-Seite angegeben. Obwohl UnoArduSim nicht die vollständige C ++ - Implementierung unterstützt, die das der GNU kompilierer zugrunde liegende Arduino.exe bietet, ist es wahrscheinlich, dass nur die fortgeschrittensten Programmierer feststellen würden, dass ein von ihnen gewünschtes C / C ++ - Element fehlt (und es natürlich immer einfache gibt) Programmierumgehungen für solche fehlenden Funktionen). Im Allgemeinen habe ich nur das unterstützt, was meiner Meinung nach die nützlichsten C / C ++ - Funktionen für Arduino-Hobbyisten und -Studenten sind - zum Beispiel: **'enum'** und **'#define'** werden unterstützt, funktionsmodul-Zeiger jedoch nicht. Obwohl benutzerdefinierte objekte (**'class'** und **'struct'**) und (die meisten) Operatorüberladungen werden unterstützt, **Mehrfachvererbung gibt es nicht**.

Da UnoArduSim ein Hochsprachen-Simulator ist, **Es werden nur C / C ++ - Anweisungen unterstützt**, **Assembler-Anweisungen sind nicht**. Da es sich nicht um eine einfache Maschinensimulation handelt, **ATmega328-Register sind für Ihren programm nicht zugänglich** zum Lesen oder Schreiben, obwohl Registerzuordnung, Weitergabe und Rückgabe emuliert werden (Sie wählen dies im Menü aus) **Optionen**).

Ab V2.6 hat UnoArduSim eingebaut automatische Unterstützung für eine begrenzte Teilmenge der Arduino-Bibliotheken zur Verfügung gestellt, wobei diese sind: **'Stepper.h'**, **'Servo.h'**, **'SoftwareSerial.h'**, **'SPI.h'**, **'Wire.h'**, **'OneWire.h'**, **'SD.h'**, **'TFT.h'** und **'EEPROM.h'** (Version 2). V2.6 stellt einen Mechanismus für 3rd Party-Bibliothek Unterstützung über dateien vorgesehen in der **'include_3rdParty'** Mappe das kann im Inneren des UnoArduSim Installationsverzeichnis zu finden.

Für jeden **'#include'** von benutzerdefinierten Bibliotheken wird UnoArduSim **nicht** Durchsuchen Sie die übliche Arduino-Installationsverzeichnisstruktur, um die Bibliothek zu finden. stattdessen du **brauchen** Kopieren Sie den entsprechenden Header (".h") und die Quelle (".cpp") datei in dasselbe Verzeichnis wie den programm datei, an dem Sie arbeiten (vorbehaltlich der Beschränkung, dass der Inhalt von irgendwelchen **'#include'** datei muss vollständig verständlich sein der UnoArduSim analysator).

Ich habe UnoArduSimV2.0 in QtCreator mit mehrsprachiger Unterstützung entwickelt und es ist derzeit nur für Fenster verfügbar TM . Portierung auf Linux oder MacOS ist ein Projekt für die Zukunft! UnoArduSim ist aus Simulatoren entstanden, die ich im Laufe der Jahre für Kurse an der Queen's University entwickelt hatte, und es wurde einigermaßen ausführlich getestet, aber es gibt bestimmt noch ein paar Fehler, die sich darin verstecken. Wenn Sie einen fehler melden möchten, beschreiben Sie ihn bitte (kurz) in einer E-Mail an unoArduSim@gmail.com und **Achten Sie darauf, Ihren vollständigen fehler-induzierenden programm-Arduino-Quellcode beizufügen** so kann ich den fehler replizieren und reparieren. Ich werde nicht auf einzelne fehler-Berichte antworten, und ich habe keine garantierten Fristen für Korrekturen in einer nachfolgenden Version (denken Sie daran, dass es fast immer Problemumgehungen gibt!).

Prost,

Stan Simmons, PhD, P. Eng.
Assoziierter Professor (im Ruhestand)
Fakultät für Elektrotechnik und Informationstechnik
Queen's University
Kingston, Ontario, Kanada







CodeBereich, Präferenzen und Editieren/Ansehen

(Nebenbei: Das unten abgebildete Beispiel fenster steht unter einem benutzerdefinierten Fenster-OS Farbschema, das eine dunkelblaue fenster-Hintergrundfarbe hat).

CodeBereich


Das **CodeBereich** Zeigt Ihren Benutzer programm an und hebt dessen ausführung hervor.

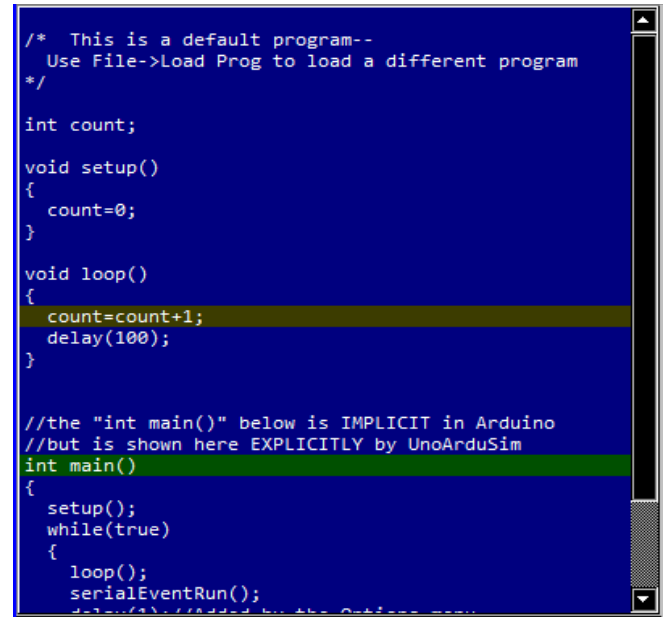
Nachdem ein geladener programm einen erfolgreichen Analysieren hat, geht die erste Zeile in 'main()' wird hervorgehoben und der programm ist bereit für ausführung. Beachten Sie, dass 'main()' wird implizit von Arduino (und von UnoArduSim) hinzugefügt **nicht** Nehmen Sie es als Teil Ihres Benutzers programm datei auf. Ausführung ist unter Kontrolle des Menüs **Ausführen** und die damit verbundenen **Werkzeugleiste** Tasten und funktionsmodul-Tastenkürzel.

Nach Schritt ausführung durch eine (oder mehr) Anweisungen (Sie verwenden können, **Tool-Bar** Tasten   , oder  die programm Linie), die ausgeführte nächst sein wird dann grün markiert - die grün markierte Zeile ist immer die nächste Zeile **bereit ausgeführte zu sein**

Ebenso, wenn ein laufender programm einen (temporären **Ausführen Dort**) haltepunkt, ausführung wird angehalten und die haltepunkt-Zeile wird hervorgehoben (und ist dann bereit für ausführung).

Wenn programm ausführung derzeit gestoppt, und Sie in der klicken **CodeBereich** fenster, die Linie, die Sie gerade geklickt wird in dunkeloliv hervorgehoben (wie im Bild gezeigt) - die nächsten-to-be-ausgeführte Linie immer grün markiert Aufenthalte (ab V2.7). Aber Sie können ausführung verursachen *vorankommen* die Linie, die Sie gerade markiert haben, indem Sie auf die Schaltfläche klicken **Ausführen**

Dort  **Werkzeugleiste** Taste. Mit dieser Funktion können Sie schnell und einfach bestimmte Zeilen in einem programm erreichen, sodass Sie anschließend zeilenweise über einen programm-Abschnitt von Interesse springen können.





```
/* This is a default program--
Use File->Load Prog to load a different program
*/

int count;

void setup()
{
  count=0;
}

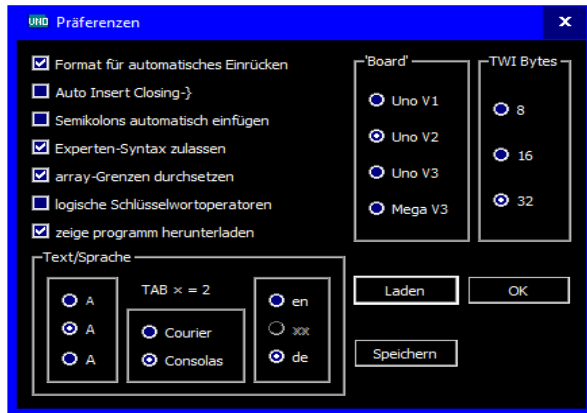
void loop()
{
  count=count+1;
  delay(100);
}

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

Wenn Ihr geladener programm irgendwelche hat '#include' dateien können Sie mit zwischen ihnen wechseln **Datei** | **Bisherige** und **Datei** | **Nächster** (mit **Werkzeugleiste** Tasten  und ).

Die Aktionen der **Finden** Menü können Sie **ENTWEDER** findet Text in der **CodeBereich** oder **VariablenBereich** (**Tool-Bar** Tasten  und  Oder Tastenkombinationen **Aufwärtspfeil** und **Pfeil nach unten**) **nach der ersten Verwendung Finden** | **Set Suche Text** oder **Tool-Bar**  . **ODER ALTERNATIV** zu **navigieren die Call-Stack** in dem **CodeBereich** (**Tool-Bar** Tasten  und  Oder Tastenkombinationen **Aufwärtspfeil** und **Pfeil nach unten**). Schlüssel **PgDn** und **PgUp** Auswahl zum nächsten / vorherigen funktionsmodul springen .

Präferenzen



Konfigurieren | Präferenzen ermöglicht Benutzern um programm und Anzeigeeinstellungen festzulegen (die ein Benutzer normalerweise bei festlegen möchte) Die nächste Sitzung). Diese können daher von einem gespeichert und geladen werden '**myArduPrefs.txt**' datei, das sich im selben Verzeichnis befindet wie das geladene 'Uno' oder 'Mega' programm ('**myArduPrefs.txt**' wird automatisch geladen, falls vorhanden).

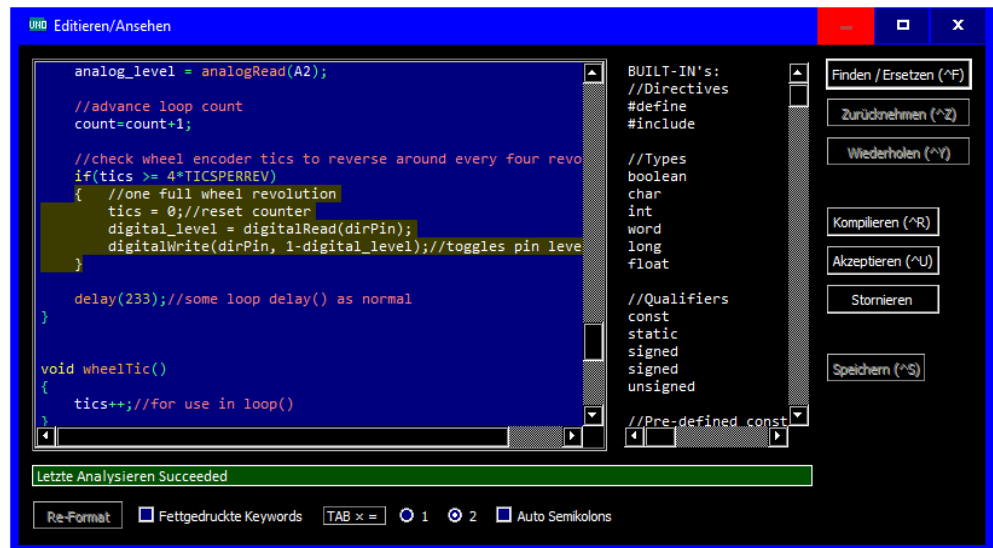
In diesem Dialogfeld können Sie zwischen zwei Schriften mit gleichem Abstand und drei Schriftgrößen sowie anderen Einstellungen wählen. Ab V2.0 ist die Sprachauswahl nun enthalten. - dazu gehört immer Englisch (**en**) sowie eine oder zwei andere Sprachen für das Gebietsschema des

Benutzers (sofern vorhanden), und eine Außerkraftsetzung basierend auf dem aus zwei Buchstaben bestehenden ISO-639-Sprachcode in der ersten Zeile des '**myArduPrefs.txt**' datei (falls vorhanden). Die Auswahl wird nur angezeigt, wenn Eine ".qm" - Übersetzung datei befindet sich im Übersetzungsordner (innen) das UnoArduSim.exe-Ausgangsverzeichnis).

Editieren/Ansehen

Durch Doppelklick auf eine beliebige Zeile in der **CodeBereich** (oder über das Menü **Datei**), ein **Editieren/Ansehen** fenster wird geöffnet, um Änderungen an Ihrem programm datei zuzulassen **aktuell ausgewählte Zeile** in dem **CodeBereich** ist markiert.

Dieser fenster verfügt über eine vollständige Bearbeitungsfunktion mit dynamischer Syntaxhervorhebung (verschiedene markieren-Farben werden für C ++ - Schlüsselwörter, Kommentare usw. verwendet). Es gibt eine optionale Hervorhebung der fettgedruckte-Syntax und eine automatische Formatierung auf Einzugsebene (vorausgesetzt, Sie haben diese mit ausgewählt) **Konfigurieren | Präferenzen**). Sie können auch bequem eingebaut funktionsmodul Anrufe auswählen (oder eingebaut '**#define**' Konstanten), die aus der bereitgestellten Listbox in Ihren programm eingefügt werden - Doppelklicken Sie einfach auf den gewünschten Listeneintrag, um ihn an der aktuellen Caret-Position zu Ihrem programm hinzuzufügen (funktionsmodul-Call variable) **Typen** dienen nur zur In formation und werden entfernt, um Platzhalter zu hinterlassen, wenn sie zu Ihrem programm hinzugefügt werden.



Mit **ALT-Pfeil nach rechts** auf Anfrage Auto-Vervollständigung Entscheidungen für eingebaut **globale variablen**, und für **Mitglied variablen** und **funktionsmodule**.

Der fenster hat **Finden** (benutzen **Strg-F**) und **Finden / Ersetzen** Fähigkeit (Verwendung **Strg-H**) . Das **Editieren/Ansehen** fenster hat **Zurücknehmen** (**Strg-Z**), und **Wiederholen** (**Strg-Y**) Schaltflächen (die automatisch angezeigt werden).

Verwerfen **alle Änderungen** Klicken Sie auf die Schaltfläche, die Sie seit dem ersten Öffnen des programm zur Bearbeitung erstellt haben **Stornieren** Taste. Akzeptieren Sie die aktueller Zustand, klicken Sie auf die Schaltfläche **Akzeptieren** und der programm empfängt automatisch einen weiteren Analysieren (und wird auf den 'Uno' oder 'Mega' heruntergeladen, wenn keine Fehler gefunden werden), und der neue Status wird im Hauptfenster von UnoArduSim fenster angezeigt **Statusleiste** .

Ein **Kompilieren** (**Strg-R**) Taste (plus eine zugehörige **Analysieren Status** Das Meldungsfeld (siehe Abbildung oben) wurde hinzugefügt, um das Testen von Bearbeitungen zu ermöglichen, ohne dass der fenster zuerst geschlossen werden muss. EIN **Speichern** (**Strg-S**) Button wurde auch als Shortcut hinzugefügt (entspricht einem **Akzeptieren** plus eine spätere separate **Speichern** vom Haupt-fenster).

An jeder **Stornieren** oder **Akzeptieren** ohne Änderungen vorgenommen, die **CodeBereich** Die aktuelle Zeile ändert sich zu **letzte Editieren/Ansehen Caret Position** , und Sie können diese Funktion verwenden, um die zu springen **CodeBereich** zu einer bestimmten Zeile (möglicherweise, um sich darauf vorzubereiten, a **Ausführen Dort**), Können Sie auch verwenden **Strg-PgDn** und **ctrl-PgUp** um zum nächsten (oder vorherigen) Leerzeilenumbruch in Ihrem programm zu springen - dies ist nützlich, um schnell zu wichtigen Stellen (wie Leerzeilen zwischen funktionsmodule) nach oben oder unten zu navigieren. Sie können auch verwenden **Strg-Startseite** und **Strg-Ende** um zum Start bzw. Ende des programm zu springen.




'Tab'-Ebene automatische Einzug Formatierung durchgeführt wird, wenn die fenster öffnet, wenn diese Option unter gesetzt wurde **Konfigurieren | Präferenzen**. Sie können jederzeit, dass die Formatierung Redo durch das Klicken auf **Re-Format** Taste (es ist nur, wenn Sie zuvor ausgewählt aktiviert die **automatisches Einrücken Preference**). Sie können auch hinzufügen oder löschen Registerkarten sich auf eine Gruppe von vorab ausgewählten aufeinander folgenden Zeilen über die Tastatur **rechter Pfeil** oder **linker Pfeil** Schlüssel - aber **automatisches Einrücken Preference aus sein müssen** zu vermeiden, Ihre eigenen Registerkarte Ebene zu verlieren.

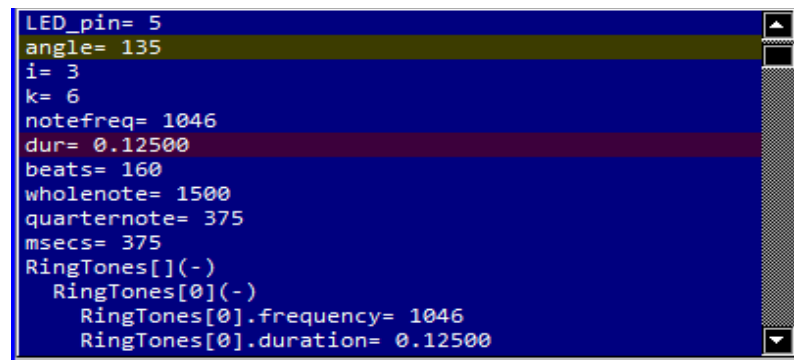
Wann **Auto-Semikolons** aktiviert ist, drücken Sie **Eingeben** um eine Zeile zu beenden, wird automatisch das Zeilenende-Semikolon eingefügt.

Und um Ihre Kontexte und geschweifte klammern besser im Auge zu behalten, klicken Sie auf a ' { ' oder ' } ' geschweifte klammer **hebt den gesamten Text zwischen diesem geschweifte klammer und seinem übereinstimmenden Partner hervor** .

VariablenBereich und Editieren/Verfolgen Variable fenster

Das **VariablenBereich** befindet sich direkt unterhalb der **CodeBereich**. Es zeigt die aktuellen Werte für jeden Benutzer global und aktiv (in-imfang) lokal variable / array / objekt im geladenen programm. Während sich Ihr programm ausführung zwischen funktionsmodule bewegt, **Der Inhalt ändert sich und spiegelt nur die lokalen variablen wider, auf die der aktuelle funktionsmodul / umfang sowie alle vom Benutzer deklarierten globalen Werte zugreifen können**. Jeder variablen, der als deklariert ist 'const' oder als 'PROGMEM' (zugeteilt an 'Flash' Memory) Werte haben, die sich nicht ändern können, und diese sind daher platzsparend **nicht angezeigt**. 'Servo' und 'SoftwareSerial' objekt-Instanzen enthalten keine nützlichen Werte und werden daher auch nicht angezeigt.

Du kannst **finden** angegeben **Text** mit seinen Text-Suchbefehlen (mit **Tool-Bar** Tasten  und  Oder Tastenkombinationen **Aufwärtspfeil** und **Pfeil nach unten**), Nach der ersten Verwendung **Finden | Set Suche** Text oder  .



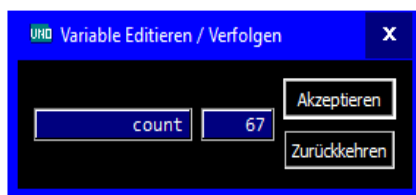
```
LED_pin= 5
angle= 135
i= 3
k= 6
notefreq= 1046
dur= 0.12500
beats= 160
wholenote= 1500
quarternote= 375
msec= 375
RingTones[0](-)
RingTones[0](-)
RingTones[0].frequency= 1046
RingTones[0].duration= 0.12500
```

Arrays und **objekte** sind entweder in gezeigt **un-erweitert** oder **erweitert** Format, entweder mit einem abschließenden Plus ' (+) ' oder minus ' (-) ' jeweils unterschreiben. Das Symbol für einen array **x** zeigt als '**x** [] ' . Um erweitern es alle Elemente des array anzuzeigen, klicken Sie einfach auf '**x** [] (+) ' in dem **VariablenBereich** . Klicken Sie auf, um zusammenziehen zu einer Ansicht ohne erweitert zurückzukehren '**x** [] (-) ' . Der un-erweitert

Standard für einen objekt 'p1' zeigt als 'p1 (+)' Um es erweitern allen Mitgliedern zu zeigen 'class' oder 'struct' Klicken Sie zum Beispiel einmal auf 'p1 (+)' in dem **VariablenBereich**. Um zusammenziehen zu einer Ansicht ohne erweitert zurückzukehren, klicken Sie einfach auf 'p1 (-)'. Wenn du *Ein-Klick auf allen Strecken markieren es in dunkeloliv* (Es kann einfach variable sein, oder das Aggregat ' (+)' oder ' (-)' Zeile eines array oder objekt oder ein einzelnes Element oder array objekt-member), dann tun **Ausführen Bis** verursacht ausführung am nächsten wieder aufnehmen und einfrieren *Schreibzugriff* überall innerhalb dieses ausgewählten Aggregats, oder dass einzelne variable Ort ausgewählt.

Beim Benutzen **Schritt** oder **Ausführen**, Aktualisierungen der angezeigten variable-Werte werden gemäß den im Menü vorgenommenen Benutzereinstellungen vorgenommen **VarAktualisieren** - Dies ermöglicht eine breite Palette von Verhaltensweisen, von minimalen periodischen Updates bis hin zu vollständigen sofortigen Updates. Reduzierte oder minimale Aktualisierungen sind nützlich, um die CPU-Auslastung zu verringern, und können erforderlich sein, um zu verhindern, dass ausführung unter den sonst zu hohen Werten in Echtzeit zurückfällt **VariablenBereich** Das fenster-Update wird geladen. Wann **Animieren** ist in Kraft, oder wenn die **Markieren Änderungen** Menüoption ausgewählt ist, ändert sich der Wert eines variable während **Ausführen** führt dazu, dass der angezeigte Wert aktualisiert wird **sofort**, und es wird hervorgehoben - dies bewirkt, dass **VariablenBereich** (falls erforderlich) zu der Zeile zu scrollen, in der sich variable und ausführung befinden, ist nicht mehr in Echtzeit !.

Wenn ausführung einfriert nach dem **Schritt**, **Ausführen Dort**, **Ausführen Bis**, oder **Ausführen** -dann- **Halt**, das **VariablenBereich** hebt den variable hervor, der dem entspricht **Adresse Ort (e), die geändert wurden** (falls vorhanden) von der **allerletzte Anweisung** während dieser ausführung (einschließlich durch variable-Deklarationsinitialisierungen). Wenn diese Anweisung **vollständig** füllte ein **objekt oder array**, das **übergeordnete (+) oder (-) Zeile** für dieses Aggregat wird hervorgehoben. Wenn stattdessen die Anweisung a Standort das ist aktuell sichtbar, dann wird es hervorgehoben. Aber wenn sich der / die geänderte (n) Ort (e) gerade darin versteckt (befinden) ein un-erweitert array oder objekt, das Aggregat **übergeordnete Zeile** bekommt eine **kursive Hervorhebung** als visueller Hinweis darauf, dass etwas in ihm geschrieben wurde - wenn Sie auf erweitern klicken, wird es dann seine **zuletzt** Geändertes Element oder Element, das hervorgehoben werden soll.



Das **Editieren/Verfolgen** fenster gibt Ihnen **die Fähigkeit, einem beliebigen variable-Wert während ausführung zu folgen**, oder zu **Ändern Sie den Wert in der Mitte von (angehalten) programm ausführung** (So können Sie testen, wie sich eine Fortsetzung dieses neuen Werts auswirken würde). **Halt** Dann zuerst ausführung **Doppelklicken Sie mit der linken Maustaste** auf dem variable, dessen Wert Sie verfolgen oder ändern möchten. Um den Wert während programm ausführung einfach zu überwachen, **Lassen Sie das**

Dialogfeld geöffnet und dann einer der **Ausführen** oder **Schritt** Befehle - sein Wert wird in aktualisiert **Editieren/Verfolgen** nach den gleichen Regeln, die Updates in der regeln **VariablenBereich**. **Zum Ändern des variable-Werts**, geben Sie den Bearbeitungsfeldwert ein und **Akzeptieren**. Fahren Sie mit ausführung fort (mit einer der Tasten **Schritt** oder **Ausführen** Befehle), um diesen neuen Wert von diesem Punkt an zu verwenden (oder Sie können **Zurückkehren** auf den vorherigen Wert).

Auf programm Laden oder Zurücksetzen Beachten Sie, dass alle *Der nicht initialisierte Wert variablen wird auf den Wert 0 zurückgesetzt, und alle nicht initialisierten Zeiger variablen werden auf 0x0000 zurückgesetzt.*

LaborBankBereich

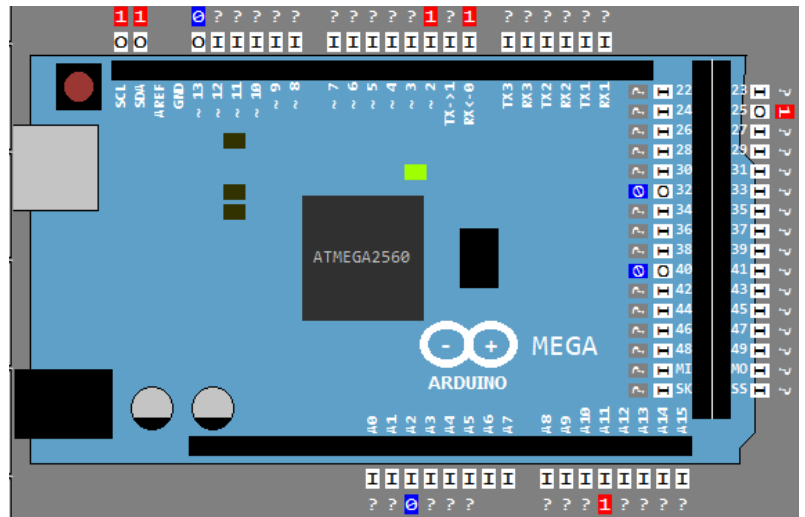
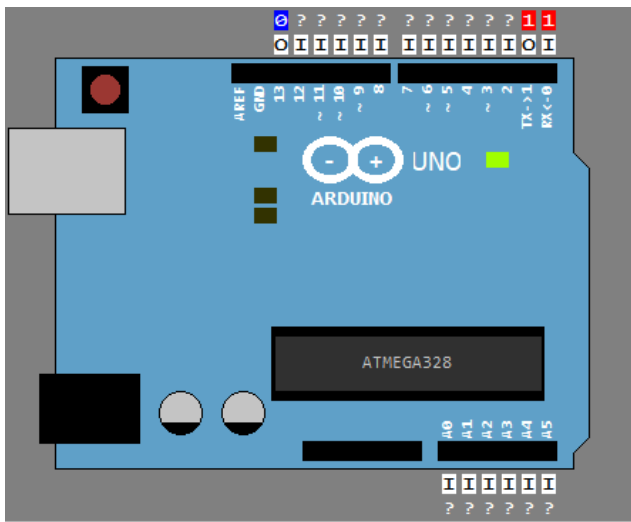
Der LaborBankBereich zeigt einen 5-Volt-'Uno' oder 'Mega'-leiterplatte, der von einem Satz 'I/O'-geräte umgeben ist, den Sie auswählen / anpassen und an den Sie den gewünschten 'Uno' oder 'Mega'-pins anschließen können.

Das 'Uno' oder 'Mega'

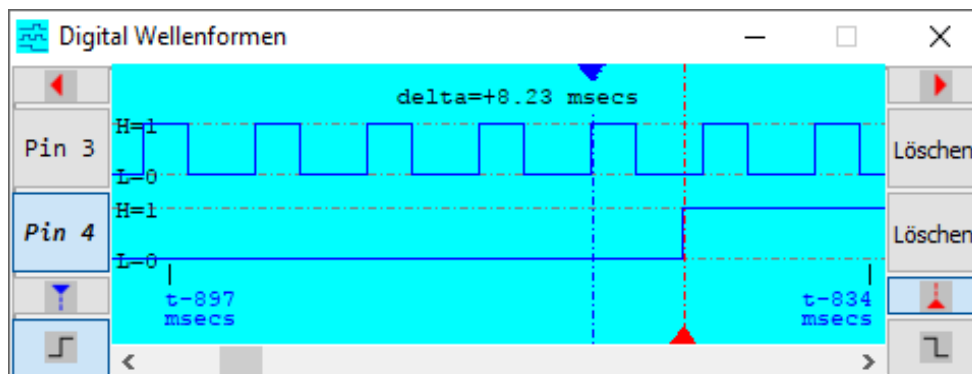
Diese ist eine Darstellung des 'Uno' oder 'Mega' leiterplatte und seiner integrierten LEDs. Wenn Sie einen neuen programm in UnoArduSim laden und ihn erfolgreich analysieren, wird dem leiterplatte ein "simulierter herunterladen" unterzogen, der die Art und Weise eines tatsächlichen leiterplatte nachahmt leiterplatte verhält sich - Sie sehen, wie der serielle RX und der TX LED blinken (zusammen mit den Aktivitäten auf pins 1 und 0) *fest verdrahtet für die serielle Kommunikation mit einem Host-Computer*). Darauf folgt sofort ein pin 13 LED-Blitz, der das Zurücksetzen von leiterplatte und (und das automatische Anhalten von UnoArduSim bei) den Beginn Ihres geladenen programm ausführung anzeigt. Sie können diese Anzeige und die damit verbundene Ladeverzögerung vermeiden, indem Sie die Auswahl aufheben **Zeigen Sie Herunterladen von Konfigurieren | Präferenzen**.

Mit dem fenster können Sie die digital-Logikpegel auf allen 20 'Uno' pins oder allen 70 'Mega' pins visualisieren ('1' auf rot für 'HIGH', '0' auf blau für 'LOW', und '?' auf grau für eine undefinierte unbestimmte Spannung) und programmierter Richtungen ('I' zum 'INPUT', oder 'O' zum 'OUTPUT'). Für pins, die mit PWM über gepulst werden 'analogWrite()', oder von 'tone()', oder von 'Servo.write()', Die Farbe ändert sich in lila und das angezeigte Symbol wird '^' .









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





Linksklick auf jedem 'Uno' öffnet pin a **Pin Digital Wellenformen** fenster, der die Vergangenheit anzeigt **eine Sekunde wert** von **Aktivität auf digital-Ebene** auf diesem pin. Sie können auf einen anderen pins klicken, um diesen zur Pin Digital Wellenformen-Anzeige hinzuzufügen (maximal 4 Wellenformen gleichzeitig).



Klicken Sie, um die Seite nach links oder rechts anzuzeigen, oder verwenden Sie die Tasten Home, PgUp, PgDn, End

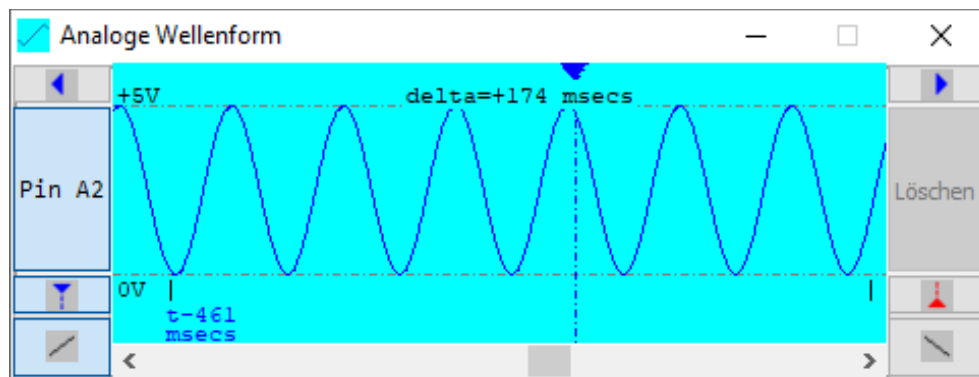
Eine der angezeigten Wellenformen ist die **aktiv pin** wellenform Dies wird dadurch angezeigt, dass die Schaltfläche "Pin" als gedrückt angezeigt wird (wie in der obigen Pin Digital Wellenformen-Bildschirmaufnahme). Sie können einen wellenform auswählen, indem Sie auf die Pin-Zifferntaste klicken, und dann die gewünschte Kantenpolarität auswählen, indem Sie auf die entsprechende Auswahl Schaltfläche für die ansteigende / abfallende Kantenpolarität klicken.  , oder  , oder mit den Tastenkombinationen **Aufwärtspfeil** und **Pfeil nach unten** . Sie können dann **springen** der aktive Cursor (entweder die blauen oder roten Cursorlinien mit der angezeigten Deltazeit) rückwärts oder vorwärts zur digital-Flanke mit der gewählten Polarität *von diesem aktiven pin* wellenform mit den Cursortasten ( ,  oder  ,  (je nachdem welcher Cursor war früher aktiviert mit  oder ) oder verwenden Sie einfach die Tastaturtasten \leftarrow und \rightarrow .

Um einen Cursor zu aktivieren, klicken Sie auf die farbige Aktivierungsschaltfläche ( oder  oben gezeigt) - *Dadurch wird die Ansicht auch zum aktuellen Standort von gescrollt dieser Cursor* . Alternativ können Sie die Aktivierung zwischen den Cursorn (mit ihren jeweils zentrierten Ansichten) über die Verknüpfung schnell abwechseln **'Tab'** Schlüssel.







Sie können **springen** der aktuell aktivierte Cursor durch **überall mit der linken Maustaste klicken** in der On-Screen-Ansicht wellenform. Alternativ können Sie entweder die rote oder die blaue Cursorzeile auswählen, indem Sie mit der rechten Maustaste darauf klicken (um sie zu aktivieren) **ziehe es auf ein neuer Ort** und loslassen. Wenn sich ein gewünschter Cursor derzeit außerhalb des Bildschirms befindet, können Sie dies tun **Klicken Sie mit der rechten Maustaste auf eine beliebige Stelle** in der Ansicht, um zu dieser neuen Position auf dem Bildschirm zu springen. Wenn beide Cursor bereits auf dem Bildschirm angezeigt werden, wechselt das Klicken mit der rechten Maustaste einfach zwischen den aktivierten Cursorn.

Verwenden Sie das Mausrad oder die Tastenkombinationen STRG-Aufwärtspfeil und STRG-Abwärtspfeil, um ZOOM IN und ZOOM OUT (der Zoom wird immer auf den AKTIVEN Cursor zentriert).

Stattdessen a **Rechtsklick auf jedem 'Uno' pin** öffnet ein **Pin Analoge Wellenform** fenster, der das anzeigt **nach einer Sekunde wert** von **Aktivität auf analoge-Ebene** auf diesem pin. Im Gegensatz zum Pin Digital Wellenformen fenster können Sie Anzeige der analoge-Aktivität auf jeweils nur einem pin.



Klicken Sie, um die Seite nach links oder rechts anzuzeigen, oder verwenden Sie die Tasten Home, PgUp, PgDn, End

Sie können **springen** das blaue oder rote Cursorlinien zum nächsten ansteigenden oder abfallenden "Steigungspunkt" mit den Vorwärts- oder Rückwärtspfeiltasten ( ,  oder  ,  , wiederum abhängig vom aktivierten Cursor, oder verwenden Sie die \leftarrow und \rightarrow Tasten) in Verbindung mit den Steigungs- / Gefälle-Auswahlstasten  ,  (Der "Steigungspunkt" tritt auf, wenn die analoge-Spannung die hohe digital-Logikpegelschwelle des ATmega pin durchläuft.) Alternativ können Sie erneut klicken, um zu springen, oder diese Cursorlinien ähnlich ihrem Verhalten im Pin Digital Wellenformen fenster ziehen

Drücken Sie **'Ctrl-S'** Innerhalb **Mit fenster können Sie den wellenform speichern (X, Y) Daten** an einen Text datei Ihrer Wahl, wo **X** ist in Mikrosekunden von der linken Seite und **Y**. ist in Bolzen.

'I/O' Geräte

Eine Reihe unterschiedlicher geräte umgibt den 'Uno' oder 'Mega' am Umfang des **LaborBankBereich** . "Small" 'I/O' geräte (von denen Sie insgesamt bis zu 16 haben dürfen) befindet sich auf der linken und rechten Seite des Bereich. "Large" 'I/O' geräte (von denen Sie insgesamt bis zu 8 zulassen) haben "aktive" Elemente und befinden sich oben und unten auf der **LaborBankBereich** . Die gewünschte Anzahl jedes verfügbaren 'I/O' gerät-Typs kann über das Menü eingestellt werden **Konfigurieren | 'I/O' Geräte** .

Kleinere 'I/O' Geräte	Big 'I/O' Geräte	
Druckknopf: 2	ServoMotor: 1	SPI Sklave: 1
Geschalteter Widerstand: 4	GleichstromMotor: 1	I2C Sklave:
Piezo Lautsprecher: 2	Schrittmotor: 1	Text LCD (SPI):
Farbige LED: 6	Pulsed Schrittmotor:	Text LCD (I2C):
4-LED Row:	Digitaler Impulsgeber: 1	Text LCD (D-4):
7-Segment LED:	Funktionsgenerator: 1	Erweiterungsport (SPI):
Pin Drahtbrücke:	SFT Serial:	Erweiterungsport (I2C):
Analoger Schieberegler: 2	SR Sklave:	Multiplexer LED (SPI):
Total (max 16): 16	1-Wire Sklave:	Multiplexer LED (I2C):
	Ein Schuss Generator:	ProgIO UNO:
		Total (max 8): 7

Jeder 'I/O' gerät verfügt über einen oder mehrere als gekennzeichnete pin-Anhänge **zwei-ziffer** pin-Nummer (00, 01, 02,... 10,11,12, 13 und danach entweder A0-A5 oder 14-19) in einem entsprechenden Eingabefeld. Für pin-Nummern 2 bis 9 können Sie einfach die einzelne ziffer eingeben - die führende 0 wird automatisch bereitgestellt, aber für pins 0 und 1 müssen Sie zuerst die führende 0 eingeben. Eingänge sind *normalerweise* auf der linken Seite eines 'I/O' gerät und Ausgänge sind *normalerweise* zur Rechten (*Platzbedarf*). Alle 'I/O' geräte reagieren direkt auf pin-Level und pin-Level-Änderungen. Daher reagieren sie entweder auf die Bibliothek funktionsmodule, die auf das angehängte pins ausgerichtet ist, oder auf programmierter '`digitalWrite()`' (für "Bit-Banged" -Operation).

Sie können mehrere geräte an dasselbe ATmega pin anschließen, solange dies kein ATmega pin erstellt **elektrische konflikt**. Ein solcher konflikt kann entweder von einem 'Uno' oder 'Mega' pin als erstellt werden '`OUTPUT`' Fahren gegen einen stark leitenden (niederohmigen) angeschlossenen gerät (z. B. Fahren gegen einen 'FUNCGEN'-Ausgang oder einen 'MOTOR') **Enc** Ausgang) oder durch zwei miteinander verbundene geräte (z. B. einen 'PULSER'- und einen 'PUSH'-Taster, die an demselben pin angeschlossen sind). Ein solches konflikt wäre in einer realen Hardware-Implementierung katastrophal und wird daher nicht zugelassen und dem Benutzer über ein Popup-Meldungsfeld angezeigt.

Über das Dialogfeld kann der Benutzer die Typen und Nummern des gewünschten 'I/O' geräte auswählen. In diesem Dialogfeld können Sie auch **Speichern** 'I/O' geräte zu einem Text datei und / oder **Laden** 'I/O' geräte aus einem zuvor gespeicherten (oder bearbeiteten) Text datei (**einschließlich aller pin-Verbindungen und anklickbarer Einstellungen sowie aller eingegebenen Bearbeitungsfelder**).

Beachten Sie, dass ab Version 1.6 die Werte in den Eingabefeldern Periode, Verzögerung und Impulsbreite im jeweiligen IO geräte mit dem Suffix 'S' (oder 's') versehen werden können. Das zeigt an, dass sie sein sollten **skaliert** nach der Position eines globalen '`I/O ____S`' Schieberegler, der im Main fenster angezeigt wird **Werkzeugleiste**. Wenn Sie den Regler ganz nach rechts bewegen, beträgt der Skalierungsfaktor 1,0 (Einheit) und Wenn sich der Schieberegler ganz links befindet, beträgt der Skalierungsfaktor 0,0 (vorbehaltlich der von jedem einzelnen 'I/O' gerät erzwungenen Mindestwerte). Sie können mehr als einen Bearbeitungsfeldwert skalieren **gleichzeitig** Verwenden Sie diesen Schieberegler. Mit dieser Funktion können Sie den Schieberegler während der Ausführung ziehen zum einfachen Emulieren sich ändernder Impulsbreiten, Perioden und Verzögerungen für die angeschlossenen 'I/O' geräte.

Der Rest dieses Abschnitts enthält Beschreibungen für jeden gerät-Typ.

Einige davon geräte **Unterstützung der Skalierung der eingegebenen Werte** Verwenden Sie den Schieberegler am Haupt-fenster **Werkzeugleiste**. Wenn der gerät-Wert den Buchstaben 'S' als Suffix hat, wird sein Wert mit einem Skalierungsfaktor (zwischen 0,0 und 1,0) multipliziert, der durch die Schieberegler-Daumen-Position bestimmt wird, vorbehaltlich der Mindestwertbeschränkung für gerät (1.0 ist ganz rechts, 0.0 ist ganz links) --siehe '`I/O ____S`' unter jedem der unten aufgeführten

Schläuche geräte.

'Serial' Monitor ('SERIAL')

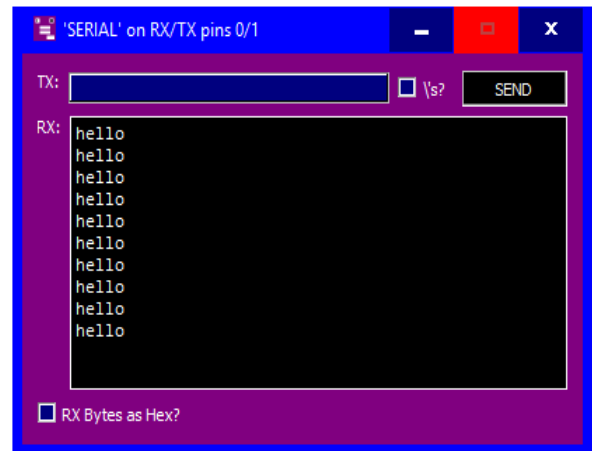


Dieser 'I/O' gerät ermöglicht die ATmega-Hardware-vermittelte serielle Ein- und Ausgabe (über den 'Uno' oder 'Mega'-USB-Chip) auf 'Uno' oder 'Mega' pins 0 und 1. Der baudrate wird über die Dropdown-Liste unten eingestellt - den ausgewählten baudrate **muss passen** der Wert, den Ihr programm an den übergibt '`Serial.begin()`' funktionsmodul für einwandfreies Senden / Empfangen. Die serielle Kommunikation ist auf 8 Datenbits, 1 Stopbit und kein Paritätsbit festgelegt. Du darfst **trennen** (leer) **aber nicht ersetzen** TX pin 00, aber nicht RX pin 01.

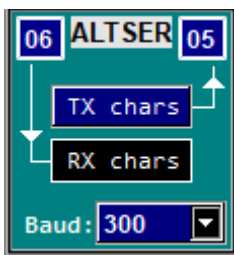
Um Tastatureingaben an Ihren programm zu senden, geben Sie ein oder mehrere Zeichen in das obere Feld (TX-Zeichen) fenster und dann ein traf die **'Enter'** Tastaturtaste . (Zeichen werden kursiv geschrieben, um darauf hinzuweisen, dass die Übertragung begonnen hat) - oder, wenn sie bereits ausgeführt wird, werden hinzugefügte eingegebene Zeichen kursiv dargestellt. Sie können dann die '`Serial.available()`' und '`Serial.read()`' funktionsmodule, um die Zeichen in der Reihenfolge zu lesen, in der sie in den pin 0-Puffer eingegangen sind (das am weitesten links stehende eingegebene Zeichen wird zuerst gesendet). Formatierte Text- und Zahlenausdrücke oder nicht formatierte Bytewerte können durch Aufrufen des Arduino an die untere Konsolenausgabe (RX-Zeichen) fenster gesendet werden '`print()`', '`println()`', oder '`write()`' funktionsmodule.

Zusätzlich, **Ein größerer fenster zum Einstellen / Anzeigen von Sende- und Empfangszeichen kann durch Doppelklicken (oder Rechtsklicken) geöffnet werden** 'SERIAL' gerät .

Dieser neue fenster verfügt über ein größeres Bearbeitungsfeld für TX-Zeichen und eine separate 'Send'-Schaltfläche, mit der die TX-Zeichen an den 'Uno' oder 'Mega' gesendet werden können (bei pin 0). Es gibt auch eine Checkbox-Option, um mit Backslash versehene Zeichenfolgen neu zu interpretieren, wie z '`\n`' oder '`\t`' für nicht rohe Anzeige.



Wechseln Seriennummer ('ALTSER')

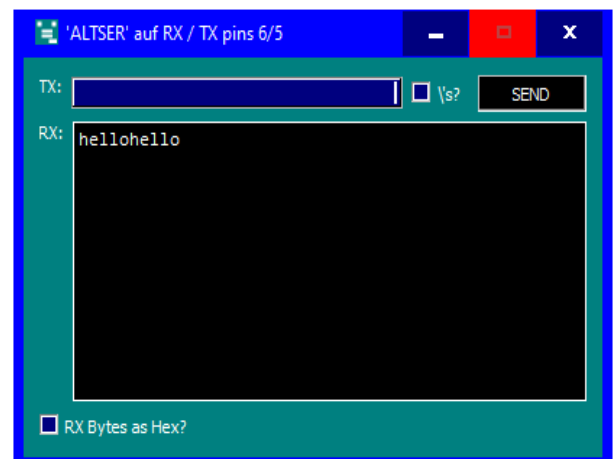


Dieser 'I/O' gerät ermöglicht die Bibliothek oder alternativ die serielle Eingabe und Ausgabe des Benutzers "Bit-Banged" für jedes Paar von 'Uno' oder 'Mega' pins, das Sie ausfüllen möchten (**ausser für** pins 0 und 1, die für Hardware bestimmt sind '`Serial`' Kommunikation). Ihr programm muss eine haben '`#include <SoftwareSerial.h>`' Zeile oben, wenn Sie die Funktionalität dieser Bibliothek nutzen möchten. Wie beim 'SERIAL' gerät wird der baudrate für 'ALTSER' über die Dropdown-Liste unten eingestellt. Der ausgewählte baudrate muss mit dem Wert übereinstimmen, den Ihr programm an den übergibt '`begin()`' funktionsmodul für ordnungsgemäßes Senden / Empfangen. Die serielle Kommunikation ist auf 8 Datenbits, 1

Stopbit und kein Paritätsbit festgelegt.

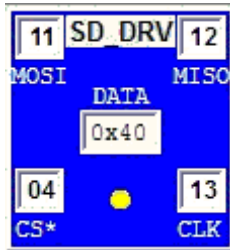
Auch wie bei der Hardware 'SERIAL' , **Ein größerer fenster für Sende- und Empfangseinstellung / -anzeige kann durch Doppelklicken (oder Rechtsklicken) auf den ALTSER gerät geöffnet werden** .

Beachten Sie, dass im Gegensatz zur Hardware-Implementierung von '`Serial`' Es ist keine vorgesehen TX-Puffer, der von internen ATmega-Interrupt-Vorgängen unterstützt wird (nur ein RX-Puffer) Das '`write()`' (oder '`print()`') Anrufe werden blockiert (das heißt, Ihr programm wird erst fortgesetzt, wenn sie abgeschlossen sind).



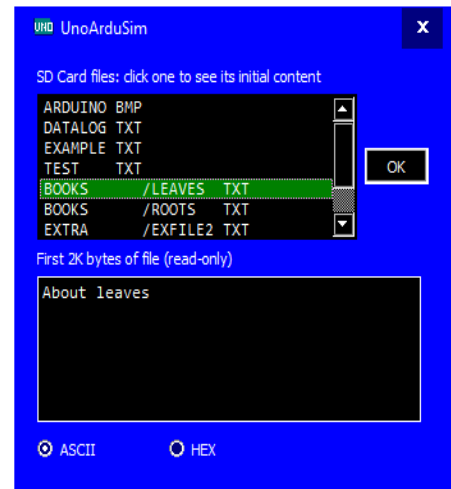
SD-Laufwerk ('SD_DRV')

Diese 'I/O' gerät ermöglicht Bibliotheks-Software-vermittelte (aber **nicht** "Bit-Banged") datei Eingabe- und Ausgabeoperationen auf dem 'Uno' oder 'Mega' **SPI** pins (Sie können wählen, welche **CS *** pin, den Sie verwenden werden). Dein programm kann das einfach `#include <SD.h>` Linie in der Nähe der Spitze, und Sie können verwenden `<SD.h>` funktionsmodule ODER direkt anrufen `SdFile` funktionsmodule selbst.



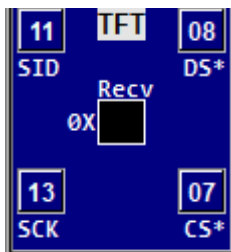
Ein größeres fenster mit Verzeichnissen und dateien (und Inhalten) kann durch Doppelklicken (oder Rechtsklicken) auf das 'SD_DRV' gerät geöffnet werden . Der gesamte Festplatteninhalt ist **geladen von** ein **SD** Unterverzeichnis im geladenen programm-Verzeichnis (falls vorhanden) unter `'SdVolume::init()'` , **und spiegelt sich zu** das gleiche **SD** Unterverzeichnis auf datei `'close()'` , `'remove()'` und weiter `'makeDir()'` und `'rmDir()'` .

Während der SPI-Übertragung blinkt ein gelbes LED, und 'DATA' zeigt das letzte 'SD_DRV' an **Antwort** Byte. Alle SPI-Signale sind genau und können in a angezeigt werden **Wellenform fenster**.



TFT Bildschirm ('TFT')

Diese 'I/O' gerät emuliert einen Adafruit™ TFT-Display der Größe 1280by-160 Pixel (in seiner nativen Rotation = 0 ist, aber wenn die 'TFT.h' Bibliothek, `'TFT.begin()'` Initialisierung Sätze für die Drehung = 1, die eine „Landschaft“ Ansicht von 160-für-Pixel 128 gibt). Sie können diese gerät antreiben durch die funktionsmodule von dem anruf `'TFT.h'` Bibliothek (die `#include <TFT.h>` erste erfordert), oder Sie können den SPI-System verwenden Sie eigene Bytefolge antreiben sie zu senden.

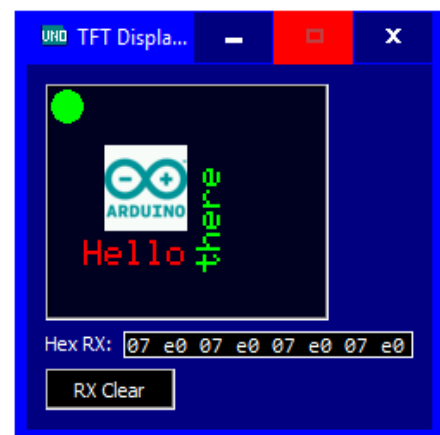


Der TFT ist immer verbunden zu 'SPI' pins 'MOSI' (für 'SID') und 'SCK' (für 'SCK') - das nicht geändert werden kann. Die 'DS*' pin ist für Daten / Befehl auswählen ('LOW' wählt Daten-Modus) und die 'CS*' pin ist die aktive-low chip-auswählen,

Es gibt keine Zurücksetzen pin zur Verfügung gestellt, so dass Sie nicht ein Hardware-Reset tun können Diese gerät durch einen pin niedrigen Fahr (wie der `'TFT::begin()'` funktionsmodul versucht

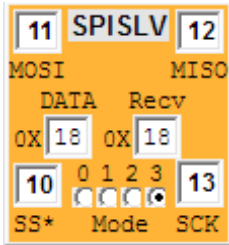
zu tun, wenn Sie eine gültige 'reset' pin Zahl bestanden haben als dritter Parameter an die `'TFT(int cs, int ds, int rst)'` Konstrukteur). Die gerät hat jedoch eine versteckte Verbindung mit dem Leitungssystem Zurücksetzen, so dass es sich jedes Mal zurücksetzt, klicken Sie auf die Haupt UnoArduSim Zurücksetzen Symbol in der Symbolleiste oder die 'Uno' oder 'Mega' leiterplatte Reset-Taste ..

Durch **Doppelklick** (oder **Rechtsklick**) Zu diesem gerät wird eine größere fenster dass die vollständige 160-für-Pixel-LCD-Anzeige 128, zusammen mit dem zuletzt empfangenen 8 Bytes zu zeigen, geöffnet wird (siehe unten)



Konfigurierbares SPI Sklave ('SPISLV')

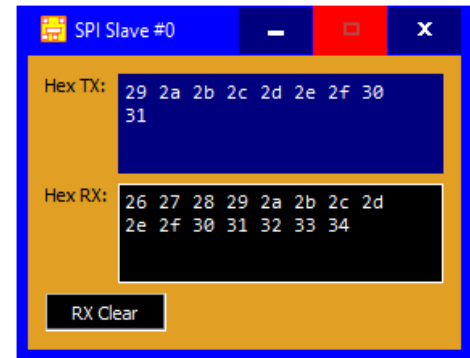
Dieser 'I/O' gerät emuliert einen SPI-Slave mit ausgewähltem Modus und aktivem Low **SS *** ("Slave-Select") pin steuert die **MISO** Ausgang pin (wenn **SS *** ist hoch, **MISO** ist nicht angetrieben). Ihr programm muss eine haben `'#include <SPI.h>'` Linie, wenn Sie die Funktionalität des eingebaut SPI Arduino objekt und Bibliothek verwenden möchten. Alternativ können Sie auch Ihr eigenes "Bit-Banged" erstellen. **MOSI** und **SCK** signalisiert antreiben diesen gerät.



Der gerät erkennt Kantenübergänge an seinen **CLK** Eingabe entsprechend dem gewählten Modus (`'MODE0'` , `'MODE1'` , `'MODE2'` , oder `'MODE3'`), die passend zum programmierter SPI-Modus Ihres programm gewählt werden muss.

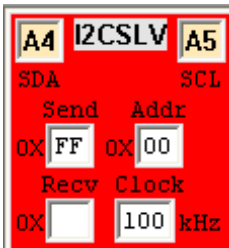
Durch Doppelklicken (oder Rechtsklicken) auf den gerät können Sie einen größeren Begleiter fenster öffnen das statt erlaubt y Sie

müssen den 32-Byte-Maximalpuffer ausfüllen (um SPI geräte zu emulieren, die automatisch ihre Daten zurückliefern) und die letzten 32 empfangenen Bytes anzeigen (alle als Hex-Paare). **Beachten Sie, dass** Das nächste TX-Pufferbyte ist Wird nur automatisch an 'DATA' gesendet **nach dem** ein voller `'SPI.transfer()'` hat vervollständigt!



Zweileiter I2C Sklave ('I2CSLV')

Dieser 'I/O' gerät emuliert nur a *Slave-Modus* gerät. Dem gerät kann eine I2C-Busadresse zugewiesen werden, indem ein Zwei-Hex-ziffer-Eintrag in seinem 'Addr'-Eingabefeld verwendet wird (er antwortet nur I2C Bustransaktionen mit der zugewiesenen Adresse). Der gerät sendet und empfängt Daten über seinen Open-Drain (nur Pull-Down) **SDA** pin und reagiert auf das Bustaktsignal an seinem Open-Drain (nur Pull-Down) **SCL** pin. Obwohl der 'Uno' oder 'Mega' der Busmaster sein wird, der für die Erzeugung der Daten verantwortlich ist **SCL** Signal wird dieser Slave gerät auch ziehen **SCL** low während der Low Phase, um die Low-Zeit des Busses auf eine der internen Geschwindigkeit entsprechende Zeit zu verlängern (die in der 'Clock' Edit-Box eingestellt werden kann).

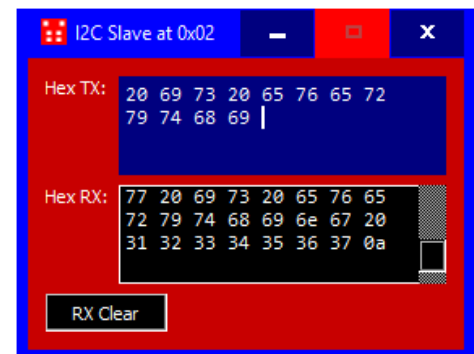


Ihr programm muss eine haben `'#include <Wire.h>'` Linie, wenn Sie die Funktionalität des `'TwoWire'` Bibliothek zur Interaktion mit diesem gerät. Alternativ können Sie auch Ihre eigenen Bit-Banged-Daten und Taktsignale für antreiben, diesen Slave gerät, erstellen.

Ein einzelnes Byte für die Rückübertragung an den 'Uno' oder 'Mega'-Master kann in das 'Send'-Bearbeitungsfeld gesetzt werden, und ein einzelnes (zuletzt empfangenes) Byte kann in seinem (schreibgeschützten) 'Recv'-Bearbeitungsfeld angezeigt werden. **Beachten Sie, dass**

Der 'Send'-Eingabefeldwert spiegelt immer die **Nächster** Byte für die Übertragung von diesem internen Datenpuffer des gerät.

Durch Doppelklicken (oder Rechtsklicken) auf den gerät können Sie einen größeren Begleiter fenster öffnen Stattdessen können Sie einen FIFO-Puffer mit maximal 32 Byte füllen (um TWI geräte mit dieser Funktionalität zu emulieren) und (maximal 32) Byte der zuletzt empfangenen Daten anzeigen (als zwei- hex-ziffer Anzeige von 8 Bytes pro Zeile). Die Anzahl der Zeilen in diesen beiden Eingabefeldern entspricht der gewählten TWI-Puffergröße (die mit ausgewählt werden kann) **Konfigurieren | Präferenzen**). Dies wurde als Option seit dem Arduino hinzugefügt `'Wire.h'` Bibliothek verwendet **fünf** Solche RAM-Puffer enthalten in ihrem Implementierungscode RAM-Speicher, was teuer ist. Durch Bearbeiten der Arduino-Installation `'Wire.h'` datei, um die definierte Konstante zu ändern `'BUFFER_LENGTH'` (und auch den Begleiter bearbeiten `'utility/twi.h'` datei zu ändern TWI buffer length) beides soll statt 16 oder 8 ein Benutzer sein **könnte** Reduzieren Sie den RAM-Speicher-Overhead von das 'Uno' oder 'Mega' in einer gezielten **Hardware-Implementierung** - UnoArduSim spiegelt daher diese reale Möglichkeit wider **Konfigurieren | Präferenzen** .

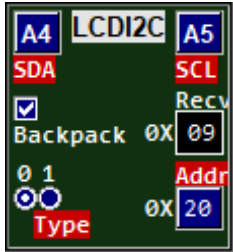


Text LCD I2C ('LCDI2C')

Diese 'I/O' gerät emuliert eine 1,2, 4-line O4 Zeichen-LCD, in einem von drei Modi:

- a) Rucksack Typ 0 (Adafruit style Port Expander mit Hardware mit I2C Busadresse 0x20-0x27)
- b) Rucksack Typ 1 (DFRobot style Port Expander mit Hardware mit I2C Busadresse 0x20-0x27)
- c) kein Rucksack (nativen Modus integriert I2C-Schnittstelle mit I2C Busadresse 0x3C-0x3F)

Unterstützung von Bibliothekscode für jeden gerät-Modus wurde in der 'include_3rdParty' Ordner Ihrer UnoArduSim Installationsverzeichnis: 'Adafruit_LiquidCrystal.h' , 'DFRobot_LiquidCrystal.h' , und 'Native_LiquidCrystal.h' , beziehungsweise.



Die gerät kann jede I2C-Bus-Adresse zugewiesen werden, unter Verwendung eines Zwei-hex-ziffer Eintrags in seiner 'Addr' Edit-Box (es wird nur reagieren, I2C Bustransaktionen seine zugewiesene Adresse beteiligt). Die gerät erhält Busadresse und Daten (und antwortet mit ACK = 0 oder NAK = 1) auf seinem Open-Drain (pull-down-only) **SDA** pin. Sie können nur Schreib LCD-Befehle und DDRAM Daten - Sie **kann nicht** Rücklesen von Daten aus den schriftlichen DDRAM Standorten ..



Doppelklick oder **Rechtsklick** den LCD-Monitor zu öffnen fenster, von dem Sie auch die Bildschirmgröße und den Zeichensatz festlegen.

Text LCD SPI ('LCDSPI')

Diese 'I/O' gerät emuliert eine 1,2, o4 4-line Zeichen-LCD, in einem von zwei Modi:

- a) Rucksack (SPI-Port-Expander Adafruit Stil)
- b) kein Rucksack (einheitlicher Modus integriert SPI Schnittstelle - wie unten gezeigt)

Unterstützung von Bibliothekscode für jeden gerät-Modus wurde in der 'include_3rdParty' Ordner Ihrer UnoArduSim Installationsverzeichnis: 'Adafruit_LiquidCrystal.h' ,, und 'Native_LiquidCrystal.h' , beziehungsweise.



Pin 'SID' ist in serielle Daten, 'SS' ist die aktive-low-gerät wählen, 'SCK' ist die Uhr pin und 'RS' ist der Daten / Befehls pin. Sie können nur Schreib LCD-Befehle und DDRAM Daten (alle SPI Transaktionen schreibt sind) - Sie **kann nicht** Rücklesen von Daten aus den schriftlichen DDRAM Standorten.

Doppelklick oder **Rechtsklick** den LCD-Monitor zu öffnen fenster, von dem Sie auch die Bildschirmgröße und den Zeichensatz festlegen.

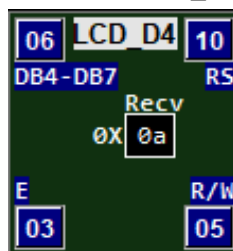


Text LCD D4 ('LCD_D4')

Diese 'I/O' gerät emuliert eine 1,2, o4 4-line Zeichen-LCD ein 4-Bit-Parallel-Bus-Schnittstelle aufweist. Datenbytes geschrieben / gelesen in **zwei Halbe** auf seinen 4-Daten pins 'DB4-DB7' (wo das Bearbeitungsfeld enthält die *niedrigste seiner 4 aufeinanderfolgenden pin Zahlen nummeriert*), - Daten über abfallende Flanken auf der 'E' getaktet werden (Freigabe) pin, mit Daten durch die 'R/W' pin und LCD-Daten / Befehls-Modus durch die 'RS' pin gesteuert.

Unterstützung von Bibliothekscode hat innerhalb der zur Verfügung gestellt worden, 'include_3rdParty' Ordner Ihrer UnoArduSim Installationsverzeichnis:

'Adafruit_LiquidCrystal.h' , , und 'Native_LiquidCrystal.h' beide arbeiten.

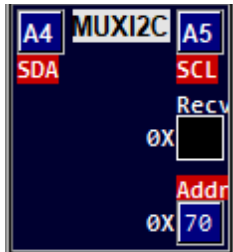


Doppelklick oder **Rechtsklick** den LCD-Monitor zu öffnen fenster, von dem Sie auch die Bildschirmgröße und den Zeichensatz festlegen.



Multiplexer LED I2C ('MUXI2C')

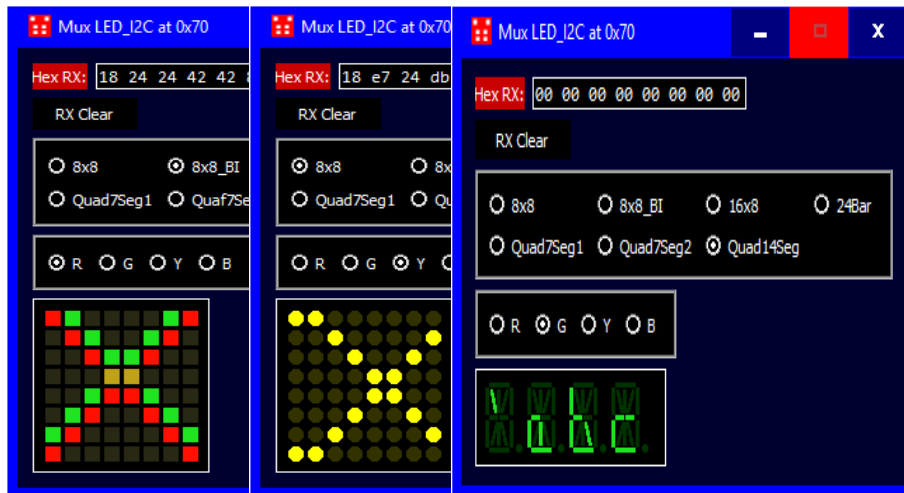
Dies 'I/O' gerät emuliert eine I2C-Schnittstelle HT16K33 Controller (h aving I2C Busadresse 0x70-0x77) zu welchem einer von mehreren verschiedenen Arten von Multiplex LED Displays angeschlossen werden:



- a) 8x8 oder 16x8 LED array
- b) 8x8 Bicolor-LED array
- c) 24-bi-color-LED bar
- d) zwei Arten von 4-ziffer 7-Segmentanzeigen
- e) eine 4-ziffer 14-Segment-Anzeige alphanumerische

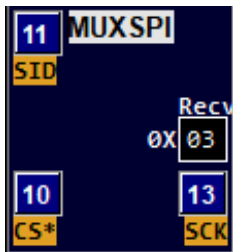
Alle sind durch die unterstützten 'Adafruit_LEDBackpack.h' Code vorgesehen innerhalb des 'include_3rdParty' Mappe:

Doppelklick (Oder Rechtsklick) zu öffnen eine größere fenster zur Auswahl und Ansicht einer der mehreren farbigen LED Displays.

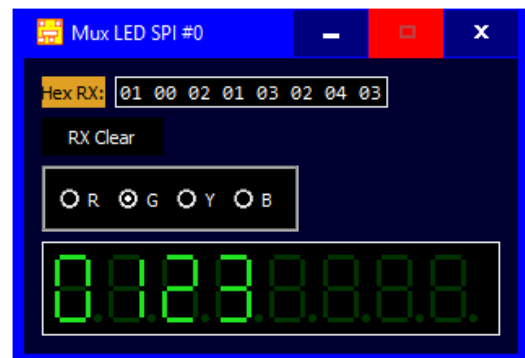


Multiplexer LED SPI ('MUXSPI')

Ein gemultiplextes-LED-Controller basierend auf der MAX6219, mit Stütz 'MAX7219.h' Code vorgesehen innerhalb des 'include_3rdParty' Ordner antreiben bis zu acht 7-Segment-Ziffern.

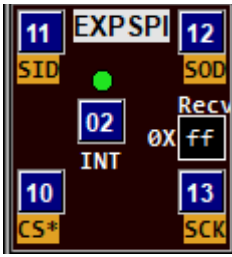


Doppelklick (Oder Rechtsklick) zu öffnen eine größere fenster sehen der farbige 8-ziffer 7-Segment- Anzeige.



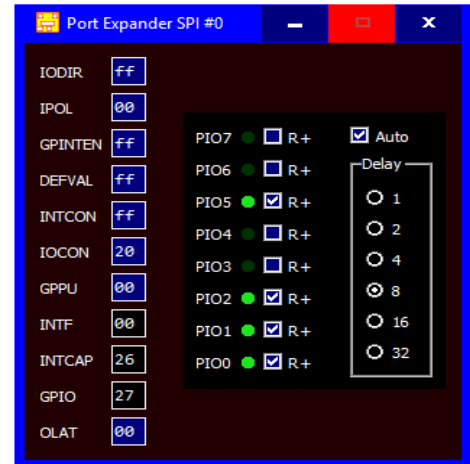
Erweiterungsport SPI ('EXPSPi')

Ein 8-bit Port-Expander basierend auf der MCP23008, mit Stütz 'MCP23008.h' Code innerhalb des 'include_3rdParty' vorgesehen Mappe. Sie können zu MCP23008 Register schreiben und lesen Sie die GPIO pin zurück Ebene. Interrupts können auf jeder GPIO pin Änderung aktiviert werden - ein ausgelöster Interrupt die 'INT' pin antreiben.



Doppelklick (Oder Rechtsklick) öffnen **eine größere fenster** sehen das 8 GPIO Portleitungen und die angebrachten Pull-up-Widerstände. Sie können Klimmzüge manuell ändern, indem Sie oder einen Zähler anhängen, die sie in regelmäßigen Abständen in einer

Aufwärtszählungs Weise verändern wird. Die Rate, mit der die Zählung Inkrementen durch die Verkleinerungs Verzögerung bestimmt wird Faktor, der durch den Benutzer ausgewählt (ein 1x Faktor entspricht ein Inkrement etwa alle 30 Millisekunden, höhere Verzögerungsfaktoren ergeben eine langsamere Aufwärtszählungsrate)

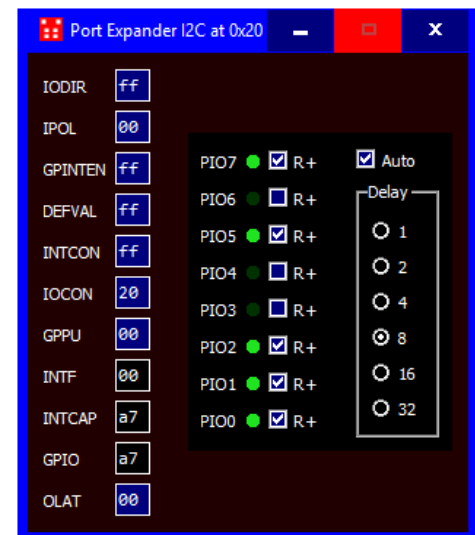


Erweiterungsport I2C ('EXPI2C')

Ein 8-bit Port-Expander basierend auf der MCP23008, mit Stütz 'MCP23008.h' Code vorgesehen innerhalb des 'include_3rdParty' Mappe. Fähigkeiten entsprechen den 'EXPSPi' gerät.

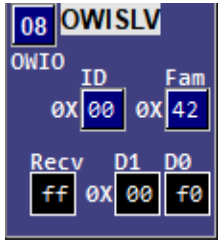


Doppelklick (Oder Rechtsklick) zu öffnen eine größere fenster wie fro der 'EXPSPi' gerät.



'1-Wire' Sklave ('OWISLV')

Dieser 'I/O' gerät emuliert einen kleinen Satz von '1-Wire'-Bus geräte, der mit pin OWIO verbunden ist. Sie können einen '1-Wire'-Bus (mit einem oder mehreren dieser Slaves '1-Wire' geräte) auf dem 'Uno' oder 'Mega' pin Ihrer Wahl erstellen. Diese gerät-Kabine kann über die 'OneWire.h' Bibliothek funktionsmodule nach Platzierung einer '#include <OneWire.h>' Linie oben auf Ihrem programm. Alternativ können Sie für diesen gerät auch Bit-Banged-Signale auf OWIO verwenden (obwohl dies sehr schwierig ist, ohne einen elektrischen konflikt zu verursachen - ein solcher konflikt ist auch bei Verwendung des gerät noch möglich) 'OneWire.h' funktionsmodule, aber solche konflikte werden in UnoArduSim) gemeldet.

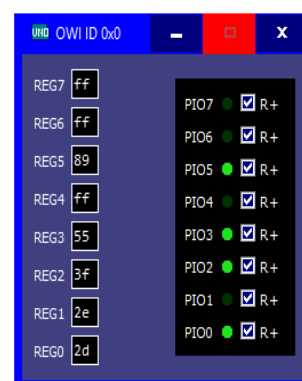
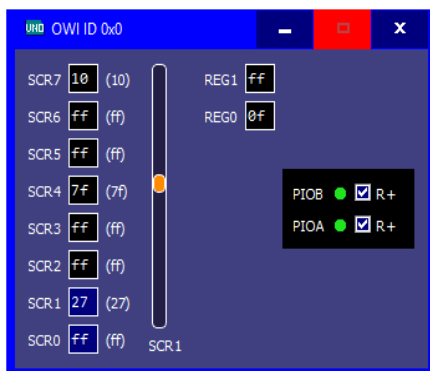


Jeder echte OWISLV gerät muss ein eindeutiges 8-Byte-i (64-Bit!) Haben. interne Seriennummer - in UnoArduSim wird dies durch den Benutzer vereinfacht, indem ein kurzes 1-Byte-hexadezimalen bereitgestellt wird 'ID' Wert (der bei Laden / Hinzufügen von gerät standardmäßig sequenziell zugewiesen wird), plus 'Fam' Familiencode für diesen gerät. UnoArduSim erkennt einen kleinen Satz von Familiencodes ab V2.3 (0x28, 0x29, 0x3A, 0x42), der den Temperatursensor abdeckt, und Parallel-E / A (PIO) geräte (ein nicht erkannter Familiencode macht den gerät zu einem generischen 8-Byte-Notizblock gerät mit generischem Sensor.

Wenn die gerät-Familie keine PIO-Register hat, werden diese registriert **D0** und **D1** vertreten. Die ersten beiden Scratchpad-Bytes, ansonsten repräsentieren sie den PIO "Status" -Register (aktuelle pin-Pegel) und PIO pin-Latch-Datenregister.

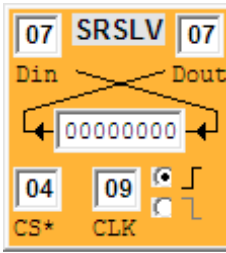
Durch **Doppelklicken** (oder **Rechtsklick**) beim gerät eine größere **OWIMonitor** fenster wird geöffnet. Von diesem größeren fenster aus können Sie alle gerät-Register überprüfen, die Scratchpad-Positionen SCR0 und SCR1 mithilfe von Bearbeitungen und einem Schieberegler ändern (SCR0 und SCR1 entsprechen ebenfalls nur **D0** und **D1** wenn kein PIO vorhanden ist) oder externe pin-PIO-Klimmzüge einstellen. Wenn SCR0 und SCR1 bearbeitet werden, merkt sich UnoArduSim diese bearbeiteten Werte als Benutzerpräferenz, die einen Anfangswert (ab Zurücksetzen) darstellt, der den vom Sensor des gerät ausgegebenen vorzeichenbehafteten Wert darstellt. der Slider wird auf 100% zurückgesetzt (ein Skalierungsfaktor von 1.0) zum Zeitpunkt der Bearbeitung. Wenn der Schieberegler anschließend bewegt wird, wird die 'signed' Der Wert in SCR1 wird entsprechend der Position des Schiebereglers verkleinert (Skalierungsfaktor von 1,0 auf 0,0). Mit dieser Funktion können Sie die Reaktion Ihres programm auf sich reibungslos ändernde Sensorwerte auf einfache Weise testen. .

Wenn Sie bei einem gerät mit PIO pins die Kontrollkästchen auf pin-Ebene aktivieren, merkt sich UnoArduSim diese aktivierten Werte als die aktuellen Klimmzüge, die extern auf den pins angewendet werden. Diese externen Pullup-Werte werden dann zusammen mit den pin-Latch-Daten (Register **D1**), um die endgültigen tatsächlichen pin-Werte zu bestimmen, und das am PIO pin angebrachte grüne LED anzuzünden oder zu löschen (das pin geht nur 'HIGH' Wenn ein externer Klimmzug angewendet wird, und die entsprechende **D1** Latch-Bit ist ein '1').



Schieberegister Sklave ('SRSLV')

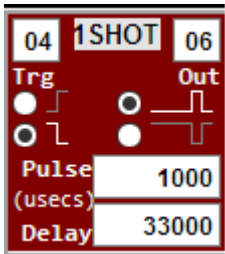
Dieser 'I/O' gerät emuliert ein einfaches Schieberegister gerät mit einem aktiven Low **SS *** ("Slave-Select") pin steuert die **'Dout'** Ausgang pin (wenn **SS *** ist hoch, **'Dout'** ist nicht angetrieben). Ihr programm könnte die Funktionalität des eingebaut SPI Arduino objekt und der Bibliothek nutzen. Alternativ können Sie auch Ihr eigenes "Bit-Banged" erstellen. **'Din'** und **CLK** signalisiert antreiben diesen gerät.



Der gerät erkennt Kantenübergänge an seinen **CLK** Eingang, der eine Verschiebung seines Registers auslöst - die Polarität der gemessenen **CLK** Die Kante kann mit einem Optionsfeld ausgewählt werden. Auf jedem **CLK** Kante (der abgetasteten Polarität) erfasst das Register seine **Lärm** Pegel in die niedrigstwertige Bit (LSB) -Position des Schieberegisters, da die verbleibenden Bits gleichzeitig um eine Position nach links in Richtung der MSB-Position verschoben werden. Wann immer **SS *** niedrig ist, liegt der aktuelle Wert in der MSB-Position des Schieberegisters angetrieben an **'Dout'**.

Generator Ein Schuss ('1SHOT')

Dieser 'I/O' gerät emuliert einen digital One-Shot, der auf seinem 'Out' einen Impuls mit der gewählten Polarität und Impulsbreite erzeugen kann pin, tritt nach einer bestimmten Verzögerung von einer an ihr empfangenen auslösenden Flanke auf **Trg** (Trigger-) Eingang pin. Sobald die angegebene Auslöseflanke empfangen wurde, beginnt timing und ein neuer Auslöseimpuls wird erst vom 'Out' erkannt Puls wurde erzeugt (und ist vollständig beendet).



Eine mögliche Verwendung dieses gerät ist das Simulieren Ultraschall-Entfernungssensoren, die als Reaktion auf einen Auslöseimpuls einen Entfernungsimpuls erzeugen. Es kann auch überall dort verwendet werden, wo Sie ein pin-Eingangssignal erzeugen möchten, das (nach der von Ihnen gewählten Verzögerung) mit einem von Ihrem programm erzeugten pin-Ausgangssignal synchronisiert ist.

'Pulse' und **'Delay'**-Werte können vom Haupt-fenster aus skaliert werden **Werkzeugleiste** 'I/O' S'-Regler für den Skalierungsfaktor durch Hinzufügen des Suffix 'S' (oder 's') an einen (oder beide).

Eine andere Verwendung für diesen gerät ist das Testen eines programm, der Interrupts verwendet, und Sie möchten sehen, was passiert, wenn a **spezifische programm Anweisung** wird unterbrochen. Trennen Sie vorübergehend den 'I/O' Gerät, den Sie mit pin 2 (oder pin 3) verbunden haben, und ersetzen Sie ihn durch einen '1SHOT' gerät, dessen 'Out' pin mit 'Uno' oder 'Mega' pin 2 (bzw. pin3) verbunden ist. Anschließend können Sie dessen 'Trg'-Eingang auslösen (vorausgesetzt, die Empfindlichkeit der ansteigenden Flanke wird dort eingestellt) durch Einfügen das Anweisungspaar { **'digitalWrite(LOW)'** , **'digitalWrite(HIGH)'** } **Grade bevor** zu der Anweisung, in der der Interrupt erfolgen soll. Stellen Sie den 1SHOT; s 'Delay' ein um den auf 'Out' erzeugten Impuls innerhalb des programm-Befehls, der auf dieses auslösende Befehlspaar folgt, zeitlich festzulegen. Beachten Sie, dass einige Anweisungen Interrupts maskieren (z. B. **'SoftwareSerial.write(byte)'** , aund so kann nicht unterbrochen werden.

Programmierbares 'I/O' Gerät ('PROGIO')



Dieser 'I/O' gerät ist eigentlich ein bloßer 'Uno' leiterplatte, den Sie programm (mit einem separaten programm) verwenden können, um einen 'I/O' gerät zu emulieren wessen Verhalten Sie vollständig definieren können. Sie können bis zu vier pins (IO1, IO2, IO3 und IO4) auswählen, die dieser Slave 'Uno' gemeinsam mit dem Master (main' Uno 'or 'Mega') in Ihrer Mitte verwendet **LaborBankBereich** . Wie bei anderen geräte wird jedes elektrische konflikt zwischen diesem 'Uno'-Slave und dem Master leiterplatte erkannt und markiert. Beachten Sie, dass alle Verbindungen sind **direkt** verdrahtet, **ausser für pin 13** (wobei ein Serien-R-1K-Widerstand zwischen den beiden pins angenommen wird, um ein elektrisches konflikt bei Zurücksetzen zu verhindern).

Ab V2.8 sind die Verbindungen zwischen Master und Slave pins **abgebildet** : Wenn der Master auch ein 'Uno' ist, ist dies **Standard** Mapping ist eine Identität (außer dass pin 1 auf pin 0 abgebildet ist und umgekehrt, um 'Serial'-Kommunikation zu ermöglichen); Wenn der Master ein 'Mega' ist, werden pins 1 und 0 erneut umgedreht. **und alles** SPI und TWI pins werden für die direkte Verbindung zwischen den entsprechenden Master- und Slave-Subsystemen zugeordnet. Diese **Standard** Mapping kann sein **überschrieben** durch Angabe in Ihrem **IODevs.txt** datei eine explizite Liste von 4 Master-pin-Nummern, die dem Namen PROGIO programm datei folgt. Wenn diese

Werte -1 sind, entspricht dies der Standardzuordnung. Für diesen gerät wurden die pin-Nummern eingegeben **sind die des Slaves 'Uno'**. Das Bild links zeigt die 4 Slaves pins, die für das SPI-System pins (SS *, MISO, MOSI, SCK) spezifiziert sind - unabhängig davon, ob er ein 'Uno' oder ein 'Mega' ist, können Sie diesen Slave als programm verwenden generischer SPI-Slave (oder Master), dessen Verhalten Sie programmgesteuert definieren können.

Durch **Doppelklicken** (oder **Rechtsklick**) Auf diesem gerät wird ein größerer fenster geöffnet, um zu zeigen, dass dieser 'Uno'-Slave einen eigenen hat **CodeBereich** und damit verbundenen **VariablenBereich**, genau wie der Master hat. Es hat auch eine eigene **Werkzeuggeste**, . Womit Sie sich beschäftigen können **Belastung** und **Kontrolle ausführung** eines Slaves programm - Die Symbolaktionen haben dieselben Tastaturkürzel wie die im Haupt-fenster. (**Laden** ist **Strg-L**, **Speichern** ist **Strg-S** usw.). Einmal geladen, können Sie ausführen von **entweder** Das Haupt-UnoArduSim fenster oder das Innere dieses Sklave-Monitors fenster - in beiden Fällen bleiben das Hauptprogramm und das Sklave-'Uno'-programm synchron zum Durchlauf der Echtzeit verriegelt, während ihre Ausführungen fortschreiten. **Um den CodeBereich zu wählen, der die Laden, Suchen und ausführung-Fokus, klicken auf Die übergeordnete fenster-Titelleiste - die nicht fokussierte CodeBereich hat dann ihre Symbolleiste Aktionen abgeblendet**.

Einige mögliche Ideen für Slave geräte, die programmierter in diesem 'PROGIO' gerät sein könnten, sind unten aufgeführt. Für die serielle, I2C- oder SPI-gerät-Emulation können Sie die entsprechende programm-Codierung mit arrays für Sende- und Empfangspuffer in der angegebenen Reihenfolge verwenden So emulieren Sie ein komplexes gerät-Verhalten:

a) Ein SPI-Master oder -Slave gerät. UnoArduSimV2.4.hat die '**SPI.h**' Bibliothek, um den Slave-Modus S {PI-Betrieb über eine optionale zu ermöglichen '**mode**' Parameter in '**SPI.begin(int mode = SPI_MASTR)**'. Ausdrücklich weitergeben '**SPI_SLV**' um den Slave-Modus zu wählen (anstatt sich auf den Standard-Master-Modus zu verlassen). Sie können jetzt auch einen Benutzeralarm funktionsmodul definieren (Nennen wir es '**onSPI**') in jedem Programm', um Bytes durch Aufrufen zu übertragen eine weitere Erweiterung hinzugefügt

'**SPI.attachInterrupt(user_onSPI)**'. **Jetzt ruf** '**rxbyte=SPI.transfer(tx_byte)**' aus deinem Inneren '**user_onSPI**' funktionsmodul löscht das Interrupt-Flag und wird **Rückkehr sofort** mit dem gerade empfangenen Byte in Ihrem variable '**rxbyte**'. Alternativ können Sie das Anhängen vermeiden ein SPI-Interrupt und stattdessen einfach anrufen '**rxbyte=SPI.transfer(tx_byte)**' aus dem Inneren Ihres Haupt-programm - dieser Anruf wird **Block ausführung** bis ein SPI-Byte übertragen wurde, und wird dann **Rückkehr** mit dem neu empfangenen Byte im Inneren '**rxbyte**'.

b) Ein Generikum **serielle 'I/O'** gerät. Sie können entweder mit dem Master leiterplatte kommunizieren '**Serial**' oder ein '**SoftwareSerial**' definiert in deinem Slave programm- für '**SoftwareSerial**' Sie müssen definiert '**txpin**' und '**rxpin**' entgegengesetzt zu denen des Matser, damit der Slave auf dem pin empfängt, auf dem der Master sendet (und umgekehrt), aber für **Nur** '**Serial**', Die 1 und 0 pins sind bereits für Sie umgedreht.

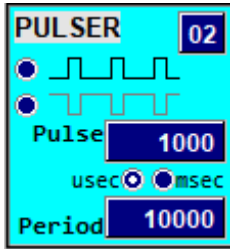
c) Ein generischer 'I2C' Master oder Slave gerät. Die Operation Sklave wurde hinzugefügt, um die Implementierung von UnoArduSim abzuschließen '**Wire.h**' Bibliothek (funktionsmodule '**begin(address)**', '**onReceive()**' und '**onRequest**' wurden nun implementiert, um den Slave-Modus zu unterstützen).

d) Ein generisches digital **Pulser**. Verwenden '**delayMicroseconds()**' und '**digitalWrite()**' ruft hinein '**loop()**'. In Ihrem 'PROGIO' programm können Sie die '**HIGH**' und '**LOW**' Intervalle einer Impulsfolge. Durch Hinzufügen eines separaten '**delay()**' ruf in deinem an '**setup()**' funktionsmodul können Sie den Start dieser Impulsfolge verzögern. Sie können sogar die Pulsbreiten variieren mit fortschreitender Zeit unter Verwendung eines Zählers variable. Sie könnten auch ein separates verwenden '**IOx**' pin als Auslöser zum Starten des timing eines emulierten '1Shot' (oder Double-Shot, Triple-Shot usw.) gerät, und Sie können die erzeugte Impulsbreite so steuern, dass sie sich im Laufe der Zeit nach Belieben ändert.

e) Ein zufälliger Signalgeber. Das ist eine Variation von a digital **Pulser** das nutzt auch anrufe nach '**random()**' und '**delayMicroseconds()**' zufällige Zeiten zu erzeugen, zu denen '**digitalWrite()**'. Ein Signal auf einem beliebigen pin, das mit dem Master geteilt wird. Mit allen vier '**IOx**' pins würde vier gleichzeitige (und eindeutige) Signale zulassen.

Digitaler Impulsgeber ('PULSER')

Dies 'I/O' gerät emuliert einen einfachen digital Puls wellenform Generator, der ein periodisches Signal erzeugt, das zu jedem gewählten 'Uno' oder 'Mega' pin angewandt werden kann.



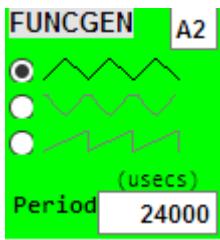
Die Periode und Impulsbreiten (in Mikrosekunden) kann so eingestellt edit-Boxen-die zulässige Mindestzeit ist 50 Mikrosekunden verwendet werden, und die minimale Impulsbreite beträgt 10 Mikrosekunden. Sie können in Mikrosekunden ('usec') und Millisekunden ('msec'), und diese Wahl wird gespeichert zusammen mit den anderen Werten zwischen timing Werten wählen, wenn Sie aus dem Konfigurieren 'Save' | I / O-Geräte Dialog.

Die Polarität kann auch ausgewählt werden: entweder positive Spitzenimpulse (0 bis 5 V) oder negative Spitzenimpulse (5 V auf 0 V).

'Pulse' und 'Period' Werte können von der Haupt skaliert werden **Tool-Bar** 'I/O_____S' skalenfaktorSchiebeRegler durch Zugabe als Suffix 'S' (oder 's') entweder eine (oder beide). Dadurch können Sie die 'Pulse' oder 'Period' Wert dann ändern **dynamisch** während ausführung.

Analoge Funktionsgenerator ('FUNCGEN')

Dieser 'I/O' gerät emuliert einen einfachen analoge wellenform-Generator, der ein periodisches Signal erzeugt, das an einen beliebigen 'Uno' oder 'Mega' pin angelegt werden kann.



Der Zeitraum (in Mikrosekunden) kann über das Bearbeitungsfeld eingestellt werden. Der minimal zulässige Zeitraum beträgt 100 Mikrosekunden. Der erzeugte wellenform kann sinusförmig, dreieckig oder sägezahnförmig gewählt werden (zum Erzeugen einer Rechteckwelle verwenden Sie stattdessen einen 'PULSER'). In kleineren Zeiträumen werden weniger Abtastwerte pro Zyklus zur Modellierung des erzeugten wellenform verwendet (nur 4 Abtastwerte pro Zyklus in Zeiträumen von 100 Mikrosekunden).

Der 'Period' Der Wert kann vom Haupt-fenster aus skaliert werden **Werkzeugeleiste** 'I/O_____S'-Regler für den Skalierungsfaktor durch Hinzufügen des Buchstabens 'S' (oder 's') als Suffix.

Schrittmotor ('STEPR')

Dieser 'I/O' gerät emuliert einen 6V bipolaren oder unipolaren Schrittmotor mit einer integrierten Treiberschaltung-Steuerung angetrieben von **entweder zwei** (auf **P1 , P2**) **oder vier** (auf **P1 , P2 , P3 , P4**) Steuersignale. Die Anzahl der Schritte pro Umdrehung kann ebenfalls eingestellt werden. Du kannst den ... benutzen 'Stepper.h' funktionsmodule 'setSpeed()' und 'step()' zu antreiben das 'STEPR'. Alternativ wird 'STEPR' *auch antworten* zu deinem eigenen 'digitalWrite()' " Bit-Banged-antreiben-Signale.



Der Motor wird sowohl mechanisch als auch elektrisch exakt modelliert. Motor-Treiberschaltung-Spannungsabfälle und unterschiedliche Widerstände und Induktivitäten 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 die pin-Impulse sehr eng gesteuert werden *nicht bekommen* der Motor läuft schrittweise - sowohl aufgrund der endlichen Anstiegszeit des Stroms als auch aufgrund der Auswirkung der Rotorträgheit. Dies

stimmt mit dem überein, was beim Antreiben eines echten Schrittmotors von einem 'Uno' oder 'Mega' mit natürlich einer geeigneten (**und erforderlich**) Motor Treiberschaltung Chip zwischen den Motorkabeln und der 'Uno' oder 'Mega'!

Ein unglücklicher fehler im Arduino 'Stepper.h' Bibliothekscode bedeutet, dass sich der Schrittmotor beim Zurücksetzen nicht in Schritt Position 1 (von vier Schritten) befindet. Um dies zu überwinden, sollte der Benutzer verwenden 'digitalWrite()' in seinem / ihrem 'setup()' Routine zum Initialisieren der Kontroll-pin-Ebenen auf die 'step(1)' Niveaus, die für die Steuerung 2-pin (0,1) oder 4-pin (1,0,1,0) geeignet sind, und lassen den Motor 10

Millisekunden auf die anfängliche gewünschte Motorposition von 12.00 Uhr Referenz fahren.

Beachten Sie, dass **Untersetzung wird nicht direkt unterstützt** aus Platzgründen, aber Sie können es in Ihrem Programm emulieren, indem Sie einen Modulo-N-Zähler variable implementieren und nur aufrufen 'step()' wenn dieser Zähler 0 erreicht (für Untersetzung um Faktor N).

AS von V2.6, diese Gerät enthält nun eine 'sync' LED (grün für synchronisierte oder rot, wenn aus durch einen oder mehrere Schritte). Auch, Zusätzlich zu der Anzahl von Schritten pro Umdrehung, zwei extra (hidden) Werte können gegebenenfalls im IODEvs.txt Datei angegeben werden, um die mechanischen last- beispielsweise angeben, Werte 20, 50, 30 Spezifiziert 20 Schritte pro Umdrehung, ein Lastträgheitsmoment 50mal der des Motorrotors selbst und eine Lastdrehmoment von 30 Prozent der vollen Motorhaltmoment.

Gepulst Schrittmotor ('PSTEPR')

Diese 'I/O' Gerät emuliert einen 6V **Mikro-Stepping** zweipolig Schrittmotor mit einem integrierten Controller Treiberschaltung angetrieben durch ein gepulster 'Step' pin, ein Aktiv-Low 'EN*' (Enable) pin und eine 'DIR' (Richtung) pin. Die Anzahl der Vollschrte pro Umdrehung kann auch direkt eingestellt wird, zusammen mit der Anzahl der Mikroschritte pro Vollschrte (1,2,4,8, oder 16). Zusätzlich zu diesen Einstellungen zwei zusätzliche (verdeckte) Werte kann optional im IODEvs.txt Datei angegeben werden, um die mechanischen last- beispielsweise angeben, Werte 20, 4, 50, 30 Spezifiziert 20 Schritte pro Umdrehung, 4 Mikroschritte pro Vollschrte, ein Lastträgheitsmoment 50mal der des Motors Rotor selbst, und ein Lastmoment von 30 Prozent des vollen Motorhaltmoment.

Sie müssen den Code schreiben antreiben die Steuer entsprechend pins.

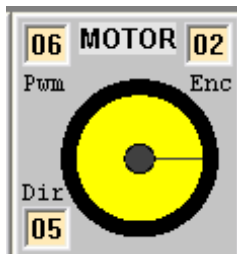


Der Motor wird genau sowohl mechanisch als auch elektrisch modelliert. Motor-Treiberschaltung Spannungsabfälle und variierende Reluktanz und Induktivität werden zusammen mit einem realistischen Trägheitsmoment in Bezug zu modellierenden Haltmoment. Der Motor hat eine Rotorwicklung modellierten Widerstand von $R = 6 \text{ Ohm}$ und eine Induktivität von $L = 6 \text{ milli-Henries}$, die eine elektrische Zeitkonstante von $1,0 \text{ ms}$ erzeugt.

diese Gerät umfasst ein gelbes 'STEP' Aktivität LED, und ein 'sync' LED (grün für die synchronisierte, oder rot, wenn sie durch einen oder mehr Schritte aus).

GleichstromMotor ('MOTOR')

Dieser 'I/O' Gerät emuliert einen 6 - Volt - 100: 1 - Getriebemotor mit integriertem Treiberschaltung - Regler angetrieben durch ein Pulsweitenmodulationssignal (auf seiner Rückseite) **Pwm** Eingang) und ein Richtungssteuersignal (an seinem Eingang) **Dir** Eingang). Der Motor hat auch einen Radgeberausgang, den treibt an hat **Enc** Ausgang pin. Sie können verwenden 'analogWrite()' bis antreiben die **Pwm** pin mit 490 Hz (bei pins 3,9,10,11) oder 980 Hz (bei pins 5,6) PWM wellenform mit einem Tastverhältnis zwischen 0,0 und 1,0 ('analogWrite()' Werte 0 bis 255). Alternativ wird 'MOTOR' auch antworten zu deinem eigenen 'digitalWrite()' " Bit-Banged-antreiben-Signale.



Der Motor wird sowohl mechanisch als auch elektrisch exakt modelliert. Berücksichtigt man die Spannungsabfälle des Motors Treiberschaltung und das realistische Leerlaufdrehmoment, ergibt sich eine volle Drehzahl von ungefähr 2 Umdrehungen pro Sekunde und ein Blockierdrehmoment von etwas mehr als 5 kg-cm (bei einem konstanten PWM-Arbeitszyklus von 1,0) mit a Gesamtträgheitsmoment Motor plus Last von $2,5 \text{ kg-cm}$. Die Motorrotorwicklung hat einen modellierten Widerstand von $R = 2 \text{ Ohm}$ und eine Induktivität von $L = 300 \text{ Mikro-Henries}$, was eine elektrische Zeitkonstante von $150 \text{ Mikrosekunden}$ erzeugt. Aufgrund der realistischen Modellierung werden Sie feststellen, dass die PWM-Impulse sehr eng sind *nicht bekommen* der Motor dreht sich - sowohl aufgrund der endlichen Stromanstiegszeit als auch

der erheblichen Ausschaltzeit nach jedem schmalen Impuls. Diese führen zusammengekommen zu einem unzureichenden Rotormoment, um das getriebefederartige Spiel unter Haftreibung zu überwinden. Die Folge ist bei der Verwendung 'analogWrite()' Bei einer Einschaltdauer von weniger als 0,125 bewegt sich der Motor nicht - dies stimmt mit dem überein, was beim Antreiben eines realen Getriebemotors von einem 'Uno' oder 'Mega' mit natürlich einer geeigneten (**und erforderlich**) Motor Treiberschaltung Modul zwischen Motor und 'Uno' oder 'Mega'!

Der emulierte Motorgeber ist ein auf einer Welle montierter optischer Unterbrechungssensor, der ein Tastverhältnis von 50% wellenform mit 8 vollständigen High-Low-Perioden pro Radumdrehung erzeugt (sodass Ihr Programm

Raddrehungsänderungen mit einer Auflösung von 22,5 Grad erfassen kann).

ServoMotor ('SERVO')

Dieser 'I/O' gerät emuliert einen positionsgeregelten PWM-angetrieben 6-Volt-Gleichstrom-Servomotor. Die mechanischen und elektrischen Modellierungsparameter für den Servobetrieb stimmen gut mit denen eines Standard-HS-422-Servos überein. Das Servo hat eine maximale Drehzahl von ca. 60 Grad in 180 Millisekunden. Wenn das Unten links Wenn das Kontrollkästchen aktiviert ist, wird der Servo zu einem **kontinuierliche Rotation** Servo mit der gleichen maximalen Geschwindigkeit, aber jetzt setzt die PWM-Impulsbreite die **Geschwindigkeit** eher als der Winkel



Ihr programm muss eine haben '`#include <Servo.h>`' vor dem Deklarieren Ihrer '`Servo`' Instanz (en) Wenn Sie die '`Servo.h`'-Bibliotheksfunktionalität verwenden möchten, z.B. '`Servo.write()`', '`Servo.writeMicroseconds()`' Alternativ reagiert 'SERVO' auch auf '`digitalWrite()`' 'Bit-Banged' -Signale. Aufgrund der internen Umsetzung von UnoArduSim sind Sie auf 6 'SERVO' geräte beschränkt.

Piezo Lautsprecher ('PIEZO')



Mit diesem gerät können Sie Signale auf einem beliebigen 'Uno' oder 'Mega' pin "abhören" und es kann eine nützliche Ergänzung zu LEDs zum Debuggen Ihres programm-Betriebs sein. Sie können auch ein bisschen Spaß beim Spielen von Klingeltönen haben, indem Sie entsprechend vorgehen '`tone()`' und '`delay()`' Anrufe (obwohl es keine Filterung des rechteckigen wellenform gibt, werden Sie keine "reinen" Noten hören).

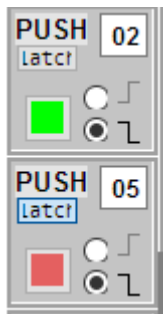
Sie können auch einen angeschlossenen 'PULSER' oder 'FUNCGEN' gerät hören, indem Sie einen 'PIEZO' an den pin, den gerät treibt an anschließen.

Schiebewiderstand ('R=1K')



Mit diesem gerät kann der Benutzer an einen 'Uno' oder 'Mega' pin entweder einen 1-k-Ohm-Pull-up-Widerstand an + 5 V oder einen 1-k-Ohm-Pull-down-Widerstand an Masse anschließen. Auf diese Weise können Sie elektrische Lasten simulieren, die einer realen Hardware gerät hinzugefügt wurden. Durch Linksklick auf den Schiebeschalter **Karosserie** Sie können die gewünschte Pullup- oder Pulldown-Auswahl umschalten. Wenn Sie einen oder mehrere dieser geräte verwenden, können Sie einen einzelnen (oder mehrere) Bit- "Code" festlegen, den Ihr programm lesen und beantworten kann.

Druckknopf ('PUSH')



Dieser 'I/O' gerät emuliert einen Schließer **momentan ODER rastend** Einpoliger Einweg-Druckknopf (SPST) mit einem Pull-Up- (oder Pull-Down-) Widerstand von 10 kOhm. Wenn für den gerät eine Übergangswahl mit ansteigender Flanke gewählt wird, werden die Tasterkontakte zwischen dem gerät pin und +5 V mit einem Pulldown von 10 kOhm gegen Masse verdrahtet. Wenn für den gerät ein Übergang mit fallender Flanke gewählt wird, werden die Tasterkontakte mit einem Pull-up von 10 kOhm auf +5 V zwischen gerät pin und Masse verdrahtet.

Durch Klicken mit der linken Maustaste oder Drücken einer beliebigen Taste schließen Sie den Tastenkontakt. Im **momentan** Im Modus bleibt es so lange geschlossen, wie Sie die Maustaste oder die Taste gedrückt halten **verriegeln** Modus (aktiviert durch Klicken auf den 'latch' Taste) bleibt es geschlossen (und in einer anderen Farbe), bis Sie die Taste erneut drücken. Kontaktprellen (für 1 Millisekunde) wird jedes Mal erzeugt, wenn Sie benutze die **Leertaste** um den Druckknopf zu drücken.

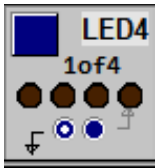
Farbige LED ('LED')



Sie können einen LED zwischen dem ausgewählten 'Uno' oder 'Mega' pin (über einen verborgenen 1-kOhm-Strombegrenzungswiderstand der Serie eingebaut) an Masse oder an +5 V anschließen. Auf diese Weise können Sie den LED aufleuchten lassen, wenn der angeschlossene 'Uno' oder 'Mega' pin eingeschaltet ist '**HIGH**' oder stattdessen wann ist es '**LOW**'.

Die LED-Farbe kann über das Bearbeitungsfeld entweder als Rot ('R'), Gelb ('Y'), Grün ('G') oder Blau ('B') ausgewählt werden.

4-LED Row ('LED4')



Sie können diese Reihe von 4 farbigen LEDs zwischen dem ausgewählten 'Uno' oder 'Mega' pins-Satz (jeder hat einen verborgenen 1-kOhm-Strombegrenzungswiderstand der Serie eingebaut) an Masse oder an +5 V anschließen. Auf diese Weise haben Sie die Wahl, ob die LEDs leuchten sollen nach oben, wenn der angeschlossene 'Uno' oder 'Mega' pin ist '**HIGH**' oder stattdessen wann ist es '**LOW**'.

Das '**1of4**' Das pin-Bearbeitungsfeld akzeptiert eine einzelne pin-Nummer **der erste von vier aufeinanderfolgenden** 'Uno' oder 'Mega' pins, der an die 4 LEDs angeschlossen wird.

Die LED-Farbe ('R', 'Y', 'G' oder 'B') ist a **versteckte Option** das kann sein **nur gewählt werden von Bearbeiten der IODevices.txt datei** (welche Sie können mit erstellen **Speichern** von dem **Konfigurieren | I/O Geräte** Dialogbox).

7-Segment LED Ziffer ('7SEG')



Sie können dieses 7-Segment-Display Ziffer LED an eine anschließen gewählter Satz von **vier aufeinanderfolgende 'Uno' oder 'Mega' pins mit dem hexadezimalen-Code** für das gewünschte angezeigte ziffer ('0' bis 'F'), und schalten Sie dieses ziffer mit dem CS * pin ein oder aus (aktiv-NIEDRIG für EIN).

Dieser gerät enthält einen eingebaut-Decoder, der die **Aktiv-Hoch** Ebenen auf den vier aufeinander folgenden '**1of4**' pins, um den anzuzeigenden angeforderten hexadezimalen ziffer zu bestimmen. Die Ebene auf der niedrigsten pin - Nummer (die in der '**1of4**' Editierfeld) repräsentiert das niedrigstwertige Bit des 4-Bit-hexadezimalen-Codes.

Die Farbe der LED-Segmente ('R', 'Y', 'G' oder 'B') ist a **versteckte Option** das kann sein **nur gewählt werden von Bearbeiten der IODevices.txt datei** Sie können mit erstellen **Speichern** von dem **Konfigurieren | I/O Geräte** Dialogbox.

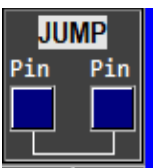
Analoger Schieberegler

Ein schiebergesteuertes 0-5-V-Potentiometer kann an einen beliebigen 'Uno' oder 'Mega' pin angeschlossen werden, um einen statischen (oder sich langsam ändernden) analoge-Spannungspegel zu erzeugen, der von diesem gelesen werden würde '**analogRead()**' als Wert von 0 bis 1023.

Verwenden Sie die Maus, um den analoge-Schieberegler zu ziehen, oder klicken Sie, um zu springen.



Pin Drahtbrücke ('JUMP')



Sie können zwei verbinden 'Uno' oder 'Mega' pins zusammen mit diesem gerät (Wenn beim Eingeben der zweiten pin-Nummer ein elektrischer konflikt erkannt wird, wird die gewählte Verbindung nicht zugelassen und der pin wird getrennt).

Dieser Jumper gerät hat eine begrenzte Nützlichkeit und ist am nützlichsten, wenn er mit Interrupts kombiniert wird, um programm zu testen, zu experimentieren und Lernzwecke. **Ab UnoArduSim V2.4 bietet die Verwendung eines 'PROGIO' gerät möglicherweise mehr Flexibilität als die**

folgenden Interrupt-angetrieben-Methoden.

Drei mögliche Verwendungen für diesen gerät sind wie folgt:

1) Sie können *Erstellen Sie eine digital-Eingabe zum Testen Ihres programm* Das timing ist komplexer als es mit einem der mitgelieferten Sets hergestellt werden kann Standard 'I/O' geräte, wie folgt:

Definieren Sie einen Interrupt funktionsmodul (nennen wir ihn '**myIntr**') und tun '**attachInterrupt(0, myIntr, RISING)**' in deinem '**setup()**'. Verbinden Sie a **Pulser** gerät bis pin2 - jetzt '**myIntr()**' wird ausführen jedes Mal, wenn a **Pulser** steigende Flanke tritt auf. Ihre '**myIntr()**' funktionsmodul kann ein Algorithmus sein, den Sie programmierter haben (unter Verwendung des globalen Zählers variablen und vielleicht sogar '**random()**') Sie können einen wellenform Ihres eigenen Designs auf jedem verfügbaren herstellen '**OUTPUT**' pin (sagen wir mal, das ist pin 9). Jetzt **SPRINGEN** pin 9 zu Ihrem gewünschten 'Uno' oder 'Mega' 'INPUT'pin, um das generierte digital wellenform auf diesen Eingang pin anzuwenden (um die Reaktion Ihres programm auf dieses wellenform zu testen). . Sie können eine Folge von Impulsen oder seriellen Zeichen oder einfach Kantenübergänge mit beliebiger Komplexität und unterschiedlichen Intervallen erzeugen. Bitte beachte, dass Wenn Ihr Haupt-programm anruft '**micros()**' (oder ruft einen funktionsmodul auf, der sich darauf verlässt), sein '**return**' Wert **wird erhöht** von der Zeit in Ihrem verbracht '**myIntr()**' funktionsmodul jedes Mal, wenn der Interrupt ausgelöst wird. Mit Aufrufen von können Sie einen schnellen Burst von genau zeitgesteuerten Flanken erzeugen '**delayMicroseconds()**' von Innerhalb '**myIntr()**' (Vielleicht, um ein Ganzes zu erzeugen **Byte** von a high-baudrate transfer) oder erzeugen Sie einfach einen Übergang pro Interrupt (vielleicht zu generieren **ein bisschen** eines Low-baudrate-Transfers) mit dem **Pulser** gerät '**Period**' passend für Ihre timing Bedürfnisse gewählt (daran erinnern, dass **Pulser** begrenzt sein Minimum '**Period**' bis 50 Mikrosekunden).

2) Sie können *Experimentieren Sie mit Subsystem-Loopbacks:*

Trennen Sie beispielsweise Ihr 'SERIAL' 'I/O' gerät TX '**00**' pin (in ein Leerzeichen ändern) und dann **SPRINGEN** 'Uno' oder 'Mega' pin '**01**' zurück zum 'Uno' oder 'Mega' pin '**00**' um eine Hardware-Schleife des ATmega zu emulieren '**Serial**' Teilsystem. Jetzt in Ihrem Test programm, innen '**setup()**' mache ein **Single** '**Serial.print()**' von a Wort oder Zeichen, und in deinem '**loop()**' Echo zurück alle Zeichen erhalten (wenn '**Serial.available()**') durch a '**Serial.read()**' gefolgt von einem '**Serial.write()**' und dann schauen, was passiert. Man könnte das ähnlich beobachten '**SoftwareSerial**' Loopback **wird versagen** (Wie im wirklichen Leben - die Software kann nicht zwei Dinge gleichzeitig tun).

Sie können es auch ausprobieren **SPI** Loopback mit a **SPRINGEN** pin 11 (MOSI) wieder mit pin 12 (MISO) verbinden.

3) Sie können *Zählen Sie die Anzahl und / oder messen Sie den Abstand bestimmter Pegelübergänge auf jedem 'Uno' oder 'Mega' Ausgang pin X* die als Ergebnis eines Komplexes auftreten Arduino-Anweisung oder Bibliothek funktionsmodul (als Beispiele: '**analogWrite()**', oder '**OneWire::reset()**', oder '**Servo::write()**') , wie folgt:





SPRINGEN pin **X** unterbrechen pin **2** und in deinem '**myIntr()**' verwenden ein '**digitalRead()**' und ein '**micros()**' Anruf, und mit gespeicherten Ebenen und Zeiten vergleichen (von vorherigen Interrupts). Sie können die Flankenempfindlichkeit für den nächsten Interrupt bei Bedarf ändern. mit '**detachInterrupt()**' und '**attachInterrupt()**' von **Innerhalb** Ihre '**myIntr()**'. Beachten Sie, dass Sie pin nicht verfolgen können Übergänge, die zu nahe beieinander liegen (näher als die gesamte ausführung - Zeit von Ihre '**myIntr()**' funktionsmodul), z. B. bei I2C- oder SPI-Übertragungen oder bei hohen baudrate '**Serial**' Übertragungen (Auch wenn Ihr Interrupt funktionsmodul die Inter-Edge timing dieser von Hardware produzierten Übertragungen nicht stören würde). Beachten Sie auch, dass softwarevermittelte Übertragungen (z '**OneWire::write()**' und '**SoftwareSerial::write()**') sind Bewusst vor Unterbrechungen geschützt (durch ihren Bibliothekscode werden vorübergehend alle Unterbrechungen deaktiviert, um timing-Unterbrechungen zu verhindern), sodass Sie mit dieser Methode keine Messungen innerhalb derer durchführen können.

Sie können stattdessen dieselben Kantenabstandsmessungen vornehmen **visuell** in einem **Digital Wellenformen** fenster, wenn Sie an einem minimalen oder maximalen Abstand über eine große Anzahl von Übergängen interessiert sind, oder wenn Sie Übergänge zählen möchten, verwenden Sie dies '**myIntr()**' -Plus- **SPRINGEN** Technik ist bequemer. Und du kannst Messen Sie zum Beispiel die Abstandsschwankungen der von programm






erzeugten Übergänge (aufgrund der Auswirkung Ihrer Software, die unterschiedliche ausführung-Pfade mit unterschiedlichen ausführung-Zeiten verwendet). eine Art zu tun von programm "Profiling".

Menüs



Datei:

<u>Laden INO oder PDE Prog (Strg-L)</u> 	Ermöglicht dem Benutzer die Auswahl eines programm datei mit der ausgewählten Erweiterung. Der programm erhält sofort einen Analysieren
<u>Editieren/Ansehen (Strg-E)</u>	Öffnet den geladenen programm zum Anzeigen / Bearbeiten.
<u>Speichern</u> 	Speichern Der bearbeitete programm-Inhalt kehrt zum ursprünglichen programm-datei zurück.
<u>Speichern Als</u>	Speichern Der bearbeitete programm-Inhalt unter einem anderen datei-Namen.
<u>Nächster ('#include')</u> 	Rückt das vor CodeBereich um den nächsten anzuzeigen '#include' datei
<u>Bisherige</u> 	Liefert die CodeBereich Anzeige zum vorherigen datei
<u>Ausgang</u>	Beendet UnoArduSim, nachdem der Benutzer daran erinnert wurde, geänderte datei (s) zu speichern.

Finden:

<u>Aufsteigen Anruf-Stapel</u> 	Zur vorherigen Anruferfunktion im Anruf-Stapel springen - der Variablenbereich passt sich dieser Funktion an
<u>Absteigen Anruf-Stapel</u> 	Zur nächsten aufgerufenen Funktion im Anruf-Stapel springen - der Variablenbereich passt sich dieser Funktion an
<u>Setze Suche Text (Strg + F)</u> 	Aktivieren Sie die Werkzeugleiste Finden Eingabefeld, um den Text zu definieren, nach dem als nächstes gesucht werden soll (und fügt das erste Wort aus der aktuell hervorgehobenen Zeile in das Feld ein) CodeBereich oder VariablenBereich wenn einer von denen hat der Fokus).
<u>inden Nächster Text</u> 	Zum nächsten Textvorkommen im springen CodeBereich (wenn es den aktiven Fokus hat) oder zum nächsten Textauftritt im VariablenBereich (wenn es stattdessen den aktiven Fokus hat).
<u>Finden Vorheriger Text</u> 	Zum vorherigen Textvorkommen im springen CodeBereich (wenn es den aktiven Fokus hat) oder auf das vorherige Textvorkommen im VariablenBereich (wenn es stattdessen den aktiven Fokus hat).

Ausführen:

<u>Schritt Hinein (F4)</u>		Schritte ausführung um eine Anweisung vorwärts oder <i>in ein genanntes funktionsmodul</i> .
<u>Schritt Über (F5)</u>		Schritte ausführung um eine Anweisung vorwärts oder <i>durch einen vollständigen funktionsmodul-Aufruf</i> .
<u>Schritt Aus (F6)</u>		Vorschüsse ausführung von <i>gerade genug, um den aktuellen funktionsmodul zu verlassen</i> .
<u>Ausführen Dort (F7)</u>		Läuft der programm, <i>Halt an der gewünschten programm-Linie</i> - Sie müssen zuerst auf eine gewünschte programm-Linie klicken, bevor Sie Ausführen Dort verwenden können.
<u>Ausführen Bis (F8)</u>		Führt den programm aus, bis ein Schreibvorgang auf den variable erfolgt, auf dem der aktuelle markieren gespeichert ist VariablenBereich (Klicken Sie auf eines, um die anfängliche markieren zu erstellen.)
<u>Ausführen (F9)</u>		Läuft der programm.
<u>Halt (F10)</u>		Stoppt programm ausführung (<i>und friert die Zeit ein</i>).
<u>Zurücksetzen</u>		Setzt den programm zurück (alle Werte von variablen werden auf 0 zurückgesetzt, und alle Zeiger von variablen werden auf 0x0000 zurückgesetzt).
<u>Animieren</u>		Führt automatisch aufeinanderfolgende programm-Zeilen aus <i>mit zusätzlicher künstlicher Verzögerung</i> und Hervorheben der aktuellen Codezeile. Echtzeitbetrieb und Geräusche gehen verloren.
<u>Zeitlupe</u>		Verlangsamt die Zeit um den Faktor 10.

Optionen:

<u>Schritt Über Tragwerke/ Operatoren</u>	Fliegen Sie direkt durch Konstruktoren, Destruktoren und die Überlastung des Bedieners funktionsmodule während eines Schrittes (dh es stoppt nicht in diesen funktionsmodule).
<u>Registerzuordnung</u>	Weisen Sie funktionsmodul-Locals den frei-ATmega-Registern statt dem Stack zu (dies führt zu einer etwas geringeren RAM-Auslastung).
<u>Fehler bei Nicht initialisiert</u>	Als Analysieren-Fehler markieren, wenn Ihr programm versucht, einen variable zu verwenden, ohne zuvor seinen Wert (oder mindestens einen Wert in einem array) initialisiert zu haben.
<u>Hinzugefügt 'loop()' 'Verzögern</u>	Fügt jedes Mal eine Verzögerung von 1000 Mikrosekunden hinzu 'loop()' wird angerufen (falls es keine anderen programm - Anrufe gibt) 'delay()' überall) - nützlich, um zu vermeiden, dass man zu weit hinter der Echtzeit zurückbleibt.
<u>Verschachtelte Interrupts zulassen</u>	Erlaube erneutes Aktivieren mit 'interrupts()' von innerhalb einer Benutzerinterrupt-Serviceroutine.

Konfigurieren:

<u>'I/O' Geräte</u>	Öffnet ein Dialogfeld, in dem der Benutzer die Art (en) und Nummern des gewünschten 'I/O' geräte auswählen kann. In diesem Dialogfeld können Sie auch Speichern 'I/O' geräte zu einem Text datei und / oder Laden 'I/O' geräte aus einem zuvor gespeicherten (oder bearbeiteten) Text datei (einschließlich aller pin-Verbindungen und anklickbaren Einstellungen und eingegebenen Werte) hinzufügen
<u>Präferenzen</u>	Öffnet ein Dialogfeld, in dem der Benutzer Einstellungen festlegen kann, einschließlich des automatischen Einrückens von programm-Quellzeilen, der Expertensyntax, der Auswahl der Schriftart schriftart, der Auswahl einer größeren Schriftgröße, der Durchsetzung von array-Grenzen und der Ermöglichung logischer Operatorschlüsselwörter, wobei programm und herunterladen angezeigt werden , Auswahl der 'Uno' oder 'Mega' leiterplatte-Version und TWI-Pufferlänge (für I2C geräte).

VarAktualisieren:

<u>Erlaube Auto (-) Zusammenziehen</u>	Lassen Sie UnoArduSim zu, dass zusammenziehen erweitert arrays / objekte anzeigt, wenn Sie in Echtzeit zurückfallen.
<u>Minimal</u>	Nur die aktualisieren VariablenBereich Anzeige 4 mal pro Sekunde.
<u>Markieren Änderungen</u>	Markieren hat während des Betriebs die variable-Werte geändert (dies kann zu einer Verlangsamung führen).

Fenster:

<u>'Serial' Monitor</u>	Schließen Sie eine serielle E / A gerät an pins 0 und 1 (falls keine vorhanden) an und ziehen Sie eine größere heraus 'Serial' TX / RX-Text fenster überwachen.
<u>Alles wiederherstellen</u>	Stelle alle minimierten Kinder fenster wieder her.
<u>Pin Digital Wellenformen</u>	Stelle einen minimierten Pin Digital Wellenformen fenster wieder her.
<u>Pin Analoge Wellenform</u>	Stelle einen minimierten Pin Analoge Wellenform fenster wieder her.

Hilfe:

<u>Schnell Hilfe Datei</u>	Öffnet das UnoArduSim_QuickHelp PDF datei.
<u>Volle Hilfe Datei</u>	Öffnet das UnoArduSim_FullHelp PDF datei.
<u>Fehler-Korrekturen</u>	Anzeigen wichtiger fehler-Korrekturen seit der vorherigen Version.
<u>Änderung / Verbesserungen</u>	Zeigen Sie wichtige Änderungen und Verbesserungen seit der vorherigen Version an.
<u>Über</u>	Zeigt die Version und das Copyright an.

'Uno' oder 'Mega' Leiterplatte und 'I/O' Geräte

Der 'Uno' oder 'Mega' und der angehängte 'I/O' geräte sind alle elektrisch genau modelliert, und Sie erhalten eine gute Vorstellung davon, wie sich Ihr programme mit der tatsächlichen Hardware verhält, und alle elektrischen pin konflikte werden markiert.

Timing

UnoArduSim führt aus schnell genug auf einem PC oder Tablet, dass es (*in den meisten Fällen*) Modell programm Aktionen in Echtzeit, **aber nur, wenn Ihr programm enthält** Zumindest einige kleine ' `delay()` ' Anrufe oder andere Anrufe, bei denen die Synchronisierung auf natürliche Weise in Echtzeit erfolgt (siehe unten).

Um dies zu erreichen, verwendet UnoArduSim einen Fenster-Callback-Timer funktionsmodul, der es ermöglicht, die Echtzeit genau zu verfolgen. Der ausführung einer Reihe von programm - Befehlen wird während eines Zeitintervalls simuliert, und Befehle, die einen längeren ausführung erfordern (wie Aufrufe von programm) ' `delay()` ') müssen möglicherweise mehrere Timer-Slices verwenden. Jede Iteration des Rückruftimers funktionsmodul korrigiert die Systemzeit mithilfe der Systemhardwareuhr, sodass programm und ausführung ständig angepasst werden, um mit der Echtzeit Schritt zu halten. *Das einzige mal ausführung bewerten **Muss in Echtzeit zurückfallen** ist, wenn der Benutzer enge Schleifen erstellt hat **ohne zusätzliche Verzögerung** , oder 'I/O' geräte sind für den Betrieb mit sehr hohen 'I/O' gerät-Frequenzen (und / oder baudrate) konfiguriert, die eine übermäßige Anzahl von pin-Änderungsereignissen und die damit verbundene Verarbeitungsüberlastung erzeugen würden. UnoArduSim kommt mit dieser Überlastung zurecht, indem einige Zeitintervalle übersprungen werden, um dies zu kompensieren, und dies verlangsamt dann den Fortschritt von programm auf **unter Echtzeit** .*

Außerdem wird programme mit großem arrays angezeigt oder weist wieder enge Schleifen auf **ohne zusätzliche Verzögerung** kann eine hohe funktionsmodul-Ruffrequenz verursachen und eine hohe erzeugen **VariablenBereich** Anzeige-Update-Last, die dazu führt, dass sie in Echtzeit zurückbleibt - UnoArduSim reduziert automatisch die variable-Aktualisierungsfrequenz, um Schritt zu halten. Wenn jedoch noch mehr Reduzierung erforderlich ist, wählen Sie **Minimal**, von dem **VarAktualisieren** Speisekarte um nur vier Aktualisierungen pro Sekunde anzugeben.

Genaue Modellierung der Sub-Millisekunden-ausführung-Zeit für jede programm-Anweisung oder -Operation **ist nicht fertig** - Für die meisten wurden nur sehr grobe Schätzungen zu Simulationszwecken angenommen. Der timing von ' `delay()` ' , und ' `delayMicroseconds()` ' funktionsmodule und funktionsmodule ' `millis()` ' und ' `micros()` ' sind alle vollkommen genau und **solange du mindestens eine der verzögerungen funktionsmodule verwendest** in einer Schleife irgendwo in Ihrem programm, **oder** Sie verwenden einen funktionsmodul, der sich auf natürliche Weise mit dem Echtzeitbetrieb verbindet (z ' `print()` ' welches an den gewählten baudrate gebunden ist), dann Die simulierte Leistung Ihres programm ist nahezu in Echtzeit (auch hier gilt, dass übermäßige hochfrequente pin-Änderungsereignisse oder übermäßige vom Benutzer zugelassene Variablen-Aktualisierungen die Leistung verlangsamen können).

Um die Wirkung der einzelnen programm-Anweisungen zu sehen, w *Henne läuft* kann es wünschenswert sein, Dinge verlangsamen zu können. Ein Zeitverlangsamungsfaktor von 10 kann vom Benutzer unter dem Menü eingestellt werden **Ausführen** .

'I/O' Gerät Timing

Diese virtuellen geräte erhalten eine Echtzeitsignalisierung von Änderungen, die an ihrem Eingang pins auftreten, und erzeugen entsprechende Ausgänge an ihrem Ausgang pins, die dann vom 'Uno' oder 'Mega' erfasst werden können - sie sind daher von Natur aus mit programm ausführung synchronisiert. Internes 'I/O' gerät timing wird vom Benutzer eingestellt (z. B. durch Auswahl von baudrate oder Taktfrequenz), und Simulatorereignisse werden eingerichtet, um den internen Echtzeitbetrieb zu verfolgen.

Geräusche

Jeder 'PIEZO' gerät erzeugt unabhängig von der Quelle solcher Änderungen einen Ton, der den am angeschlossenen pin auftretenden elektrischen Pegeländerungen entspricht. Damit die Sounds mit programm ausführung synchron bleiben, startet und stoppt UnoArduSim die Wiedergabe eines zugeordneten Soundpuffers, wenn ausführung gestartet / angehalten wird.

Ab Version 2.0 wurde der Sound für die Verwendung der Qt-Audio-API geändert - leider für den QAudioOutput 'class' unterstützt nicht das Schleifen des Sound-Puffers, um zu vermeiden, dass die Sound-Samples ausgehen (wie es bei längeren OS-Fensteroperationsverzögerungen der Fall sein kann). Daher wird der Ton jetzt nach der folgenden Regel stummgeschaltet, um die große Mehrheit der störenden Geräuschklicks und Geräuschunterbrechungen während Betriebssystemverzögerungen zu vermeiden:

Der Ton ist stummgeschaltet, solange UnoArduSim nicht der "aktive" fenster ist (außer wenn gerade ein neuer untergeordneter fenster erstellt und aktiviert wurde). **und selbst** wenn UnoArduSim der "aktive" Haupt-fenster ist, der Mauszeiger jedoch **draußen** von Hauptkundenbereich von fenster.

Beachten Sie, dass dies impliziert, dass der Ton vorübergehend ist. Stumm geschaltet wie der Mauszeiger schwebt **über ein Kind fenster**, und wird stumm geschaltet **ob Das Kind fenster wird angeklickt, um es zu aktivieren** (bis der Haupt-UnoArduSim fenster erneut angeklickt wird, um ihn wieder zu aktivieren) .

Sie können die Stummschaltung jederzeit aufheben, indem Sie auf eine beliebige Stelle im Client-Bereich des UnoArduSim-Hauptgeräts fenster klicken.

Aufgrund der Pufferung s ound hat eine Echtzeitverzögerung von bis zu 250 Millisekunden gegenüber der entsprechenden Ereigniszeit auf dem pin des angeschlossenen 'PIEZO'.

Einschränkungen und nicht unterstützte Elemente

Enthaltenes Dateien

Ein '<>' - in Klammern '#include' von '<Servo.h>', '<Wire.h>', '<OneWire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' und '<SD.h>' ist unterstützt aber diese werden nur emuliert - die aktuellen dateien werden nicht gesucht; Stattdessen ist ihre Funktionalität direkt in UnoArduSim "integriert" und gilt für die fest unterstützte Arduino-Version.

Irgendwelche zitiert '#include' (zum Beispiel von "supp.ino", "myutil.cpp", oder "mylib.h") wird unterstützt, aber alle solche dateien müssen **wohnen in der gleiches Verzeichnis als übergeordnetes programm datei**. Das enthält ihre '#include' (Es wird nicht in anderen Verzeichnissen gesucht). Das '#include' Diese Funktion kann nützlich sein, um die Menge des programm-Codes zu minimieren, der in der angezeigt wird **CodeBereich** zu jeder Zeit. Header dateien mit '#include' (dh diejenigen mit a ".h" Erweiterung) veranlasst den Simulator außerdem, den gleichnamigen datei mit a ".cpp" Erweiterung (falls diese auch im Verzeichnis des übergeordneten programm vorhanden ist).

Dynamische Speicherzuordnungen und RAM

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

Übermäßige RAM-Nutzung für variable-Deklarationen wird zur Analysieren-Zeit gekennzeichnet, und ein RAM-Speicherüberlauf wird während der programm-ausführung-Zeit gekennzeichnet. Ein Menüpunkt **Optionen** Ermöglicht die Emulation der normalen ATmega-Registerzuordnung, wie sie vom AVR kompilierer durchgeführt wird, oder die Modellierung eines alternativen Kompilierungsschemas, das nur den Stack verwendet (als Sicherheitsoption für den Fall, dass ein fehler in der Modellierung der Registerzuordnung angezeigt wird). Wenn Sie einen Zeiger zum Anzeigen des Stapelinhalts verwenden, sollte dieser genau das widerspiegeln, was in einer tatsächlichen Hardwareimplementierung angezeigt wird.

'Flash' Speicherzuordnungen

'Flash' Speicher 'byte', 'int' und 'float' variablen / arrays und der zugehörige Lesezugriff funktionsmodule werden unterstützt. Irgendein 'F()' funktionsmodul anrufen ('Flash' -Makro) von jede wörtliche Zeichenfolge ist unterstützt, aber die einzigen unterstützten 'Flash'-Speicherzeichenfolgen mit direktem Zugriff sind funktionsmodule 'strcpy_P()' und 'memcpy_P()', Um ein anderes funktionsmodule zu verwenden, müssen Sie zuerst den 'Flash'-String in ein normales RAM kopieren 'String' variable, und dann mit diesem RAM arbeiten 'String'. Wenn Sie die 'PROGMEM' variable-Modifikator Schlüsselwort, es muss erscheinen **vor** der Name variable und der

Name variable **muss auch deklariert werden** wie 'const' .

'String' Variablen

Der Einheimische 'String' Die Bibliothek wird mit wenigen sehr (und geringfügigen) Ausnahmen fast vollständig unterstützt.

Das 'String' Unterstützte Operatoren sind +, + =, <, <=, >, > =, ==, !=, und [] . Beachten Sie, dass: 'concat()' nimmt ein **Single** Argument, das ist die 'String', oder 'char', oder 'int' an das Original angehängt werden 'String' objekt, **nicht** zwei Argumente, wie fälschlicherweise auf den Arduino-Referenz-Webseiten angegeben).

Arduino-Bibliotheken

Nur 'Servo.h' . 'SoftwareSerial.h' . 'SPI.h' . 'Wire.h' . 'OneWire.h', 'Stepper.h' . 'SD.h' , 'TFT.h' und 'EEPROM.h' für die **Arduino V1.8.8** Release derzeit in UnoArduSim unterstützt. V2.6 stellt einen Mechanismus für 3rd Party-Bibliothek Unterstützung über dateien vorgesehen in der 'include_3rdParty' Mappe das kann im Inneren des UnoArduSim Installationsverzeichnis zu finden. Versuchen '#include' das "CPP" und ".h" dateien andere bis jetzt noch nicht unterstützt Bibliotheken **nicht arbeiten** wie sie Low-Level-Montageanleitung und nicht unterstützte Richtlinien und unerkannt dateien enthalten!

Zeiger

Zeiger auf einfache Typen, arrays oder objekte, werden unterstützt. Ein Zeiger kann mit einem array desselben Typs gleichgesetzt werden (z. B. 'iptr = intarray'), aber dann würde es sein *Keine anschließende arrays-Grenzprüfung* auf einen Ausdruck wie 'iptr[index]' .

Funktionsmodule kann Zeiger zurückgeben, oder 'const' Zeiger, aber jede nachfolgende Ebene von 'const' auf den zurückgegebenen Zeiger wird ignoriert.

Es gibt **keine Unterstützung** für funktionsmodul werden Anrufe durchgestellt **Benutzerdefinierte funktionsmodul-Zeiger** .

'class' und 'struct' Objekte

Obwohl Polymorphismus und Vererbung (bis zu einer beliebigen Tiefe) unterstützt werden, a 'class' oder 'struct' kann nur definiert werden, um höchstens zu haben **ein** Base 'class' (dh **mehrere**- Vererbung wird nicht unterstützt). Base-'class'-Konstruktorinitialisierungsaufrufe (über Doppelpunktnotation) in Konstruktordeklarationszeilen werden unterstützt, aber **nicht** Elementinitialisierungen mit derselben Doppelpunktnotation. Dies bedeutet, dass objekte enthalten 'const' Nicht-'static'-variablen oder Referenztyp variablen werden nicht unterstützt (diese sind nur mit angegebenen Initialisierungen von Konstruktionszeitelementen möglich).

Überladungen von Kopierzuweisungsoperatoren werden zusammen mit Verschiebungskonstruktoren und Verschiebungszuweisungen unterstützt, benutzerdefinierte objekt-Konvertierungen ("type-cast") funktionsmodule werden jedoch nicht unterstützt.

Imfang

Es gibt keine Unterstützung für die 'using' Schlüsselwort oder für 'namespace', oder für 'file' imfang. Alle nicht lokalen Deklarationen werden von der Implementierung als global angenommen.

Irgendein 'typedef', 'struct', oder 'class' Definition (dh das kann für zukünftige Erklärungen verwendet werden), muss gemacht werden **global** imfang (**lokal** Definitionen solcher Elemente in einem funktionsmodul werden nicht unterstützt.

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

Das 'unsigned' Präfix funktioniert in allen normalen rechtlichen Kontexten. Das 'const' Schlüsselwort, wenn verwendet, muss **vorausgehen** der Name variable oder funktionsmodul oder 'typedef' Name, der deklariert wird - Wenn Sie ihn nach dem Namen setzen, wird ein Analysieren-Fehler verursacht. Zum funktionsmodul-Deklarationen können nur Zeiger zurückgeben, die funktionsmodule zurückgeben 'const' erscheinen in ihrer Erklärung.

Alles UnoArduSim variablen sind 'volatile' durch die Umsetzung, so die 'volatile' Das Schlüsselwort wird in allen variable-Deklarationen einfach ignoriert. Funktionsmodule dürfen nicht deklariert werden 'volatile', noch sind funktionsmodul-Aufruf Argumente.

Das 'static' Das Schlüsselwort ist für normales variablen und für objekt-Member und Member-funktionsmodule zulässig, für objekt-Instanzen selbst jedoch ausdrücklich nicht zulässig ('class' / 'struct'), für Nicht-funktionsmodule und für alle funktionsmodul-Argumente.

Kompilierer-Richtlinien

'#include' und regelmäßig '#define' werden beide unterstützt, aber **kein Makro** '#define'. Das '#pragma' Richtlinie und bedingte Eingliederungsrichtlinien ('#ifdef', '#ifndef', '#if', '#endif', '#else' und '#elif') sind auch **nicht unterstützt**. Das '#line', '#error' und vordefinierte Makros (wie '_LINE_', '_FILE_', '_DATE_', und '_TIME_') sind auch **nicht unterstützt**.

Arduino-sprachige Elemente

Alle muttersprachlichen Arduino-Sprachelemente mit Ausnahme der zweifelhaften werden unterstützt 'goto' Eine Anweisung (die einzige vernünftige Verwendung, die ich mir vorstellen kann, ist ein Sprung (zu einer Endlosschleife für die Rettung und das sichere Herunterfahren) im Falle eines Fehlerzustands, den Ihr programm sonst nicht behandeln kann.)

C / C ++ - Sprachelemente

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

'union' ist **nicht unterstützt**.

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

Funktionsmodul Vorlagen

Benutzerdefinierte funktionsmodule, die das Schlüsselwort "template" verwenden, um Argumente vom Typ "generic" zu akzeptieren, sind **nicht unterstützt**.

Echtzeit-Emulation

Wie oben erwähnt, sind ausführung-Zeiten der vielen verschiedenen einzelnen möglichen Arduino programm-Anweisungen **nicht** genau modelliert, so dass Ihr programm eine Art dominieren muss, um mit einer Echtzeirate zu laufen 'delay()' Anweisung (mindestens einmal pro 'loop()') oder eine Anweisung, die natürlich mit Echtzeitänderungen auf pin-Ebene synchronisiert ist (z. B. 'pulseIn()', 'shiftIn()', 'Serial.read()', 'Serial.print()', 'Serial.flush()' usw.).

Sehen **Timing** und **Geräusche** Weitere Informationen zu Einschränkungen finden Sie oben.

Versionshinweise

Fehler-Korrekturen

V2.8.1- Juni 2020

- 1) Zusätzliche Leerzeilen, die durch die automatische Formatierungseinstellung entfernt wurden, führten dazu, dass die ursprünglich hervorgehobene Zeile in Editieren/Ansehen um die Anzahl der darüber entfernten Leerzeilen nach unten versetzt wurde.
- 2) 'analogRead()' akzeptierte nur die vollständige digital pin-Nummer.
- 3) Bei einem 'class', der funktionsmodul-Überladungen enthielt, die sich ausschließlich im 'unsigned'-Attribut eines funktionsmodul-Arguments unterscheiden, wurde möglicherweise die falsche Überladung funktionsmodul aufgerufen. Dies betraf LCD 'print(char/int/long)', wo stattdessen die 'unsigned'-Basis-10-Drucküberlastung funktionsmodul aufgerufen wurde, und dies führte dazu, dass LCD 'printFloat()' '46' anstelle des '.'-Dezimalpunkts druckte.
- 4) Das automatische Einfügen (nach 'Enter') eines bereits übereinstimmenden eingebaut funktionierte nach dem Zurückverfolgen nicht, um einen Tippfehler in der Zeile zu korrigieren
- 5) 'TFT'-Geräte mit einem leeren 'RS'-Pin können bei der Ausführung einen absturz verursachen

V2.8.0- Juni 2020

- 1) Wenn eine Parse-Warnung (aber kein Parse-Fehler) auftrat, konnte V2.7 versuchen, die problematische Zeile im falschen Puffer hervorzuheben (stattdessen im neuesten '# include' -Puffer), und dies würde einen stillen Absturz verursachen (sogar vorher) Das Warn-Popup wird angezeigt, wenn die Zeilennummer über dem Ende dieses Puffers '#include' liegt.
- 2) Die empfangenen Bytes in den größeren Monitorfenstern der Geräte 'I2CSLV', 'SPISLV', 'TFT', 'LCDI2C', und 'LDCSPI' wurden immer noch nicht alle korrekt gemeldet (dies hatte keinen Einfluss auf ihre Funktion).
- 3) Variablen vom Typ '**unsigned char**' wurden zum Typ befördert '**int**' in arithmetischen Ausdrücken, aber nicht ordnungsgemäß beibehalten '**unsigned**' Status, der zu einem Fehler führt '**unsigned**' resultierender Ausdruck.
- 4) Das Ändern (oder Austasten) des pin eines 'LED4' oder '7SEG' gerät von einer anfänglich gültigen pin-Einstellung konnte die oberen 3 pins nicht lösen und konnte zu ungeklärten Abstürzen führen. Außerdem wurden Änderungen in V2.7 vorgenommen, um die Handhabung aller 'LED' zu harmonisieren hat den 'LED4' gerät komplett kaputt gemacht.
- 5) Ein Fehler bei Änderungen, die für Version 2.6 zur Unterstützung von 'LOW'-Interrupts vorgenommen wurden, führte dazu, dass an pin 3 angeschlossene 'FALLING'-Interrupts die Erkennung von Änderungen auf Interrupt-Ebene auf pin 2 beschädigten.
- 6) 'SoftwareSerial' hat Benutzerinterrupts während jedes empfangenen Zeichens nicht ordnungsgemäß deaktiviert.
- 7) Wenn ein 'PROGIO'-Programm mit einem Analysefehler geladen wurde, wurde der Fehler markiert, aber dieses Programm wurde bereits durch ein Standard-'PROGIO'-Programm im 'PROGIO'-Monitorfenster ersetzt.
- 8) Seit Version V2.4 wurden pin-Änderungen an Pin 1 und Pin 0 beim Herunterladen nicht gesehen.
- 9) Das Schleifen von Lese- oder Schreibvorgängen auf einem offenen SD datei führte dazu, dass die ausführung-Sequenzierung fehlschlug, was zu einem möglichen internen UnoArduSim-Fehler aufgrund der Tiefe des imfang-Pegels führte.
- 10) Der Versuch, den 'MISO' pin auf einem 'SD_DRV' gerät zu ändern, hat den MOSI pin beschädigt.
- 11) Alle früheren Versionen konnten dies nicht warnen '**analogWrite()**' auf pins 9 oder 10 würde 'Servo' timing für alle aktiven 'Servo' geräte beschädigen.

V2.7.0- März 2020

- 1) Wenn der (Fenster-default) Thema Licht OS angenommen wurde, das **CodeBereich** in V2.6 eingeführt (statt nur ein grau markieren aus einer Systemüberschreibung) nicht die Markierungsfarbe-zeigt.
- 2) Version 2.6 versehentlich brach auto-indent-Registerkarte Formatierung in der ersten `'switch()'` konstruieren.
- 3) Die neue anruf-stapel-Navigation-Funktion in Version 2.6 eingeführt zeigten **falsche Werte** für die lokale variablen, wenn sie nicht innerhalb des aktuell ausgeführten funktionsmodul, und scheiterte mit verschachteltem Mitglied funktionsmodul Anrufen.
- 4) `'TFT :: text ()'` funktionierte, aber `'TFT :: print ()'` Funktionen funktionierten nicht (sie wurden einfach für immer blockiert). Außerdem ist `'TFT :: loadImage ()'` fehlgeschlagen, wenn `'Serial.begin ()'` früher ausgeführt wurde (was der Normalfall ist und jetzt erforderlich ist)
- 5) Version 2.6 eingeführt, um eine fehler, die den falschen Wert angezeigt für Gegenwart und Vergangenheit 'RX' Bytes für 'I2CSLV', 'SPISLV', 'TFT', 'LCDI2C' and 'LCDSPI'geräte (und dessen Monitor-fenster).
- 6) Eine Änderung in V2.4 verursacht Ausführen | Animieren viele ausgeführte Code-Zeilen zu überspringen, hervorheben.
- 7) Seit V2.4, 'SS*' oder 'CS*' auf einem 'I/O' gerät de-Behauptung sofort in der Anweisung nach einer `'SPI.transfer()'` würde dazu führen, dass gerät zum Scheitern verurteilt den übertragenen Daten-Byte zu empfangen. Darüber hinaus Byte Empfang in `'SPI_MODE1'` und `'SPI_MODE3'` wurde nicht bis zum Beginn des nächsten Bytes gesendet durch den Master (wurde abgewählt, bevor dann und das Byte vollständig verloren, wenn die gerät 'CS*') Liste steht.
- 8) In der neuen `'SPI_SLV'` Modus seit V2.4 erlaubt, `'bval = SPI.transfer()'` nur wieder den richtigen Wert für `'bval'` wenn die Byte-Übertragung war bereits fertig und warten, wenn `'transfer()'` hieß.
- 9) Das 'DATA' Bearbeitungsfeld auf 'SPISLV' geräte bekommt jetzt die Standard-0xFF, wenn es kein weiteres Bytes antworten mit.
- 10) Die Synchronisation LED Zustand war falsch für 'PSTEPR' geräte pro Vollschrift mehr als 1 Mikroschritt mit.
- 11) Elektrische konflikte verursacht durch 'I/O' geräte auf Übergänge auf dem 'SPI' Takt Umsetzen eines 'PWM' Signal oder einem `'tone'` Signal, wurden nicht berichtet, und auf unerklärliche führen könnte (beschädigt) Datenempfang.
- 12) Wenn das Intervall zwischen den Interrupts zu klein (weniger als 250 Mikrosekunden), a (fehlerhaft) in V2.4 Änderung der timing von eingebaut funktionsmodule, dass die Verwendung entweder Systemzeitgebern oder Befehlsschleifen verändert, um Verzögerungen zu erzeugen (Beispiele für jeweils `'delay()'` und `'delayMicroseconds()'`). Eine nachträgliche Änderung in V2.5 verursachte Fehlausrichtung von `'shiftOut()'` Daten- und Taktsignale, wenn eine Unterbrechung zwischen den Bits passiert.
- 13) ein eingebaut funktionsmodul automatische Vervollständigung Text über die Enter-Taste Annahme fehlgeschlagen Parametertypen aus dem Text des inserierten funktionsmodul Anruf Streifen aus.
- 14) (ausführung der neuen Interrupt-Routine versucht, aufgrund fehlerhaften) einen neuen (Benutzer-Interrupt-angetrieben) programm Laden, wenn ein zuvor ausgeführt programm hatte noch einen Interrupt während des Herunterladens zu einem Absturz könnte dazu führen, noch aus.
- 15) Objekt-Mitglied Auto-Vervollständigungen (über 'ALT'-Pfeil nach rechts) für objekte innen `'#include'` dateien sind jetzt zugänglich, sobald ihre `'#include'` datei ist erfolgreich analysiert.
- 16) Eine Erklärung funktionsmodul einen versehentlichen Raum innerhalb eines funktionsmodul Parameternamen, die verursachte eine unklare Analysieren Fehlermeldung.
- 17) Wenn ausführung in einem Modul unterscheidet sich von der Haupt programm gestoppt, die Datei | Vorherige Aktion fehlgeschlagen aktiviert sein.
- 18) Einzel zitierte geschweifte klammern (`'{'` und `'}'`) Wurden nach wie vor (fälschlicherweise) als imfang Klammern in der gezählten Analysieren, und auch Auto-Tab-indent Formatierung verwirrt ..
- 19) `'OneWire::readBytes(byte* buf, int count)'` hatte bereits Fehler sofort die angezeigten aktualisieren `'buf'` Inhalte im VariablenBereich.

20) Oktal-Latch 'OWISLV' geräte zeigte Ausgang pin Ebene, die von einem Latch-Register-Schreibzurückblieb.

V2.6.0- Jan 2020

- 1) Ein fehler in V2.3 eingeführt zu einem Absturz führte, wenn die zugegebene **Schließen** Button wurde in dem verwendeten **Finden / Ersetzen** Dialog (statt dem Ausgang Titel-Bar-Taste).
- 2) Wenn ein Benutzer programm tat '**#include**' anderer Benutzer dateien, die **Speichern** Knopf innen **Editieren/Ansehen** würde tatsächlich speichern einer modifizierten datei versagt haben, wenn es eine vorhandene Analysieren oder Ausführung Fehler innerhalb einer anderen datei Liste steht.
- 3) EIN **Stornieren** nach einer **Speichern** auch verwirrend sein könnte - aus diesen Gründen die **Speichern** und **Stornieren** Taste Funktionen geändert wurden (siehe **Änderungen und Verbesserungen**)..
- 4) Unausgeglichene Klammern innerhalb eines '**class**' Definition könnte einen Hang seit V2.5 verursachen.
- 5) Direkte logische Prüfung auf '**long**' Werte zurück '**false**' wenn keine der 16 Bits niedrigster wurden eingestellt.
- 6) UnoArduSim hatte das Markieren von einem Fehler, wenn ein Zeiger variable als Schleife variable innerhalb der Klammern einen deklarierten wurde '**for()**' Erklärung.
- 7) UnoArduSim hatte disallowing logische Tests, dass im Vergleich Zeiger auf '**NULL**' oder '**0**'.
- 8) UnoArduSim hatte Zeigerarithmetik worden disallowing eine ganze Zahl variable beteiligt (nur ganzzahlige Konstanten erlaubt worden war).
- 9) Interrupts gesetzt mit '**attachInterrupt(pin, name_func, LOW)**' hatte nur auf ein erfaßt wurde **Überleitung** zu '**LOW**'.
- 10) Wenn mehr als ein 'I2CSLV'-Gerät verwendet wird, kann ein nicht adressierter Slave spätere Busdaten als mit seiner Busadresse (oder seiner Global Call-Adresse 0x00) übereinstimmend interpretieren und so fälschlicherweise ACK signalisieren, wodurch der Bus-ACK-Pegel verfälscht wird und ein 'requestFrom()' aufgehängt wird
- 11) Passing numerischen Wert '**0**' (oder '**NULL**') Als funktionsmodul Argument einen Zeiger in einem funktionsmodul Anruf wird nun erlaubt.
- 12) Das Streifeneinschusseinzugsebene nach einer Verschachtelung '**switch()**' Konstrukte waren zu flach, wenn die 'auto-indent formatting' Wahl **Konfigurieren | Präferenzen** wurde benutzt.
- 13) Subtraktion von zwei kompatiblen Zeiger ergibt sich nun in einer Art '**int**'.
- 14) UnoArduSim erwartet hatte einen benutzerdefinierten Standardkonstruktor für ein Mitglied objekt auch wenn es nicht so erklärt worden war '**const**'.
- 15) Bei einer ausführung Pause, die gezogene Position eines 'STEPR', 'SERVO' oder 'MOTOR' Motor könnte um bis zu 30 Millisekunden der Bewegung hinter seiner tatsächlichen zuletzt berechnete Position hinken.
- 16) '**Stepper::setSpeed(0)**' war ein Absturz wegen einer Division durch Null verursacht.
- 17) Einzelne Zeile '**if()**', '**for()**', und '**else**' Konstrukte nicht mehr Ursache ein zu viele Auto-Einzug Registerkarten.

V2.5.0- Oktober 2019

- 1) Ein fehler eingeführt in V2.4 brach 'SD' Karteninitialisierung (ein Absturz verursacht).
- 2) Unter Verwendung des 'SPI'-Subsystem in den neuen '**SPI_SLV**' Modus funktionierte nicht richtig in '**SPI_MODE1**' und '**SPI_MODE3**'.
- 3) Auto-Vervollständigung Pop-ups (wie von 'ALT-right=arrow' angefordert) wurden für funktionsmodule mit objekt Parameter; die Pop-ups Liste jetzt gehören auch (Basen-) Klasse geerbten Elemente und Auto-Vervollständigungen jetzt auch erscheinen für '**Serial**'.
- 4) Seit V2.4 der Rückgabewert für '**SPI.transfer()**' und '**SPI.transfer16()**' falsch war, wenn ein Benutzer Interruptroutine während dieser Übertragung gebrannt.

- 5) In V2.4, schnelle periodische Wellenformen wurden als eine längere Dauer als ihre tatsächliche Dauer gezeigt offensichtlich, wenn bei höheren Zoom angesehen.
- 6) der Konstruktor `'File::File(SdFile &sdf, char *fname)'` so funktioniert nicht, `'File::openNextFile()'` (Die an diesem Konstruktor beruht) funktioniert auch nicht.
- 7) UnoArduSim wurde fälschlicherweise eine Analysieren Fehler auf objekt-variablen erklärt und objekt-Rückkehr funktionsmodule, erklärt als `'static'`.
- 8) Zuweisungsanweisungen mit einem `'Servo'`, `'Stepper'`, oder `'OneWire'` objekt variable auf der LHS, und ein objekt-Rückkehr funktionsmodul oder Konstruktor auf der RHS, verursacht eine interne Fehlzählung der Anzahl der zugeordneten objekte, was zu einem eventuellen Absturz.
- 9) Boolesche Tests auf objekte vom Typ `'File'` wurden immer wiederkehr `'true'` selbst wenn die datei war nicht geöffnet.

V2.4 - Mai 2019

- 1) Ausführen Bis-Überwachungspunkte können fälschlicherweise durch ein Schreiben auf ein benachbartes variable (1 Byte niedrigere Adresse) ausgelöst werden.
- 2) Die Auswahl der Baudrate konnte erfasst werden wenn sich die Maus in einem `'SERIAL'` oder `'SFTSER'` befand gerät Baud Dropdown-Listbox (Auch wenn keine Baudrate angeklickt wurde).
- 3) Seit V2.2 sind serielle Empfangsfehler mit einer Baudrate von 38400 aufgetreten.
- 4) Eine in V2.3 vorgenommene Änderung hat dazu geführt **'SoftwareSerial'** zuweilen fälschlicherweise einen deaktivierten Interrupt melden (und so beim ersten Empfang fehlschlagen).
- 5) Eine in V2.3 vorgenommene Änderung führte dazu, dass SPISLV geräte Hex-Werte falsch interpretierte, die direkt in das `'DATA'`-Bearbeitungsfeld eingegeben wurden.
- 6) Eine in V2.3 vorgenommene Änderung verursachte SRSLV geräte manchmal fälschlicherweise und stillschweigend ein elektrisches konflikt an ihrem `'Dout'` pin erkennen und so eine pin-Zuordnung dort verbieten. Wiederholte Versuche, einen pin an `'Dout'` anzuschließen, können zu einem Absturz führen, sobald der gerät entfernt wurde.
- 7) Der Versuch, einen `'LED4'` gerät nach pin 16 anzubringen, würde einen Absturz verursachen.
- 8) Ein fehler wird durch schnelle wiederholte Aufrufe von ausgelöst `'analogWrite(255)'` zum **'MOTOR'** Kontrolle im Benutzer programm verursacht resultierend **'MOTOR'** Geschwindigkeiten falsch sein (viel zu langsam).
- 9) Seit V2.3 konnte dem SRSLV geräte aufgrund einer fehlerhaften elektrischen konflikt-Erkennung kein `'Dout'` pin zugewiesen werden (und daher eine pin-Zuordnung nicht zulassen).
- 10) SPI-Slaves setzen nun ihre Sender- und Empfängerlogik zurück, wenn ihre **'SS'** pin geht **'HIGH'**.
- 11) Berufung `'Wire.h'` funktionsmodule `'endTransmission()'` oder `'requestFrom()'` Wenn Interrupts derzeit deaktiviert sind, wird jetzt ein ausführung-Fehler generiert (`'Wire.h'` müssen Interrupts aktiviert sein, um zu arbeiten).
- 12) `'Ctrl-Home'` und `'Ctrl-End'` funktionieren jetzt wie erwartet in Editieren/Ansehen.
- 13) Das `'OneWire'` Bus-Befehl `0x33 ('ROM_READ')` funktionierte nicht und hängte den Bus auf.

V2.3 - Dezember 2018

- 1) Ein in V2.2 eingeführtes fehler machte es unmöglich, den Wert eines array-Elements darin zu bearbeiten **Editieren/Verfolgen**.
- 2) Seit Version 2.0 ist der Text in der **'RAM free'** Das Steuerelement der Symbolleiste war nur sichtbar, wenn ein dunkles Fenster-OS-Design verwendet wurde.
- 3) Auf **Datei | Laden** und auf I / O gerät datei **Laden**, die datei Filter (wie **'*.ino'** und **'*.txt'**) funktionierten nicht - dateien aller Arten wurden stattdessen angezeigt.
- 4) Der "modifizierte" Zustand eines datei ging dabei verloren **Akzeptieren** oder **Kompilieren** in einem nachfolgenden **Datei | Editieren/Ansehen wenn keine weiteren Änderungen vorgenommen wurden**

(**Speichern** wurde deaktiviert, und es gab keine automatische Aufforderung zu **Speichern** auf programm **Ausgang**).

- 5) Betreiber `'/=` und `'%='` gab nur korrekte Ergebnisse für `'unsigned'` linke Seite variablen
- 6) Die ternäre Bedingung `'(bval) ? s1:s2'` gab ein falsches Ergebnis, wenn `'s1'` oder `'s2'` war ein lokaler Ausdruck anstelle eines variable.
- 7) Funktionsmodul `'noTone()'` wurde korrigiert, um zu werden `'noTone(uint8_t pin)'`.
- 8) Ein langjähriger fehler verursachte einen Absturz nach einem **Zurücksetzen** wenn dieser Zurücksetzen in der Mitte eines funktionsmodul ausgeführt wurde, der mit einem fehlenden Parameter aufgerufen wurde (und daher einen Standardinitialisierungswert erhielt).
- 9) Mitgliedsausdrücke (zB `'myobj.var'` oder `'myobj.func()'`) haben das nicht geerbt `'unsigned'` Eigentum ihrer rechten Seite (`'var'` oder `'func()'`) und konnten daher nicht direkt mit anderen verglichen oder kombiniert werden `'unsigned'` Typen - eine zwischenzeitliche Zuordnung zu einem `'unsigned'` variable wurde zuerst benötigt.
- 10) UnoArduSim bestand darauf, dass die Definition eines funktionsmodul einen Parameter hatte mit einem Standardinitialisierer, dass der funktionsmodul einen früheren prototyp deklariert hat.
- 11) Anrufe nach `'print(byte bvar, base)'` fälschlicherweise befördert `'bvar'` zu einem `'unsigned long'`, und so zu viele Ziffern ausgedruckt.
- 12) `'String(unsigned var)'` und `'concat(unsigned var)'` und Betreiber `'+=(unsigned)'` und `'+(unsigned)'` falsch erstellt `'signed'` stattdessen Streicher.
- 13) Ein 'R=1K' gerät, geladen von einem **IODevices.txt** datei mit Position `'U'` wurde fälschlicherweise mit dem Schieberegler (immer) in die **entgegengesetzte Position** aus seiner wahren elektrischen Position.
- 14) Versuch, sich auf die Standardeinstellung zu verlassen `'inverted=false'` Argument bei der Deklaration von a `'SoftwareSerial()'` objekt verursachte einen Absturz und ging vorbei `'inverted=true'` funktionierte nur, wenn der Benutzer programm eine nachfolgende tat `'digitalWrite(txpin, LOW)'` um zunächst den erforderlichen Leerlauf festzustellen `'LOW'` Ebene auf der **TX** pin.
- 15) 'I2CSLV' geräte reagierte nicht auf Änderungen in den pin-Bearbeitungsfeldern (die Standardeinstellungen für A4 und A5 blieben bestehen).
- 16) 'I2CSLV' und 'SPISLV' geräte haben Teilbearbeitungen nicht erkannt und korrigiert, als die Maus ihre Ränder verlassen hat
- 17) Pin-Werte für geräte, die auf eine SPISLV oder SRSLV folgten, wurden nicht ordnungsgemäß in der gespeichert **IODevs.txt** datei als Hexadezimalzahl.
- 18) Beim Versuch, mehr als ein SPISLV gerät MISO an pin 12 anzuschließen, trat immer ein elektrischer Konflikt-Fehler auf.
- 19) Umschalten eines pin von `'OUTPUT'` zurück zu `'INPUT'` -Modus Fehler beim Zurücksetzen des pin-Daten-Latch-Pegels auf `'LOW'`.
- 20) Verwenden `'sendStop=false'` bei Anrufen nach `'Wire.endTransmission()'` oder `'Wire.requestFrom()'` einen Fehler verursacht.
- 21) UnoArduSim erlaubte zu Unrecht a `'SoftwareSerial'` Empfang gleichzeitig mit a `'SoftwareSerial'` Getriebe.
- 22) Variablen deklariert mit `'enum'` Typ konnte nach seiner Deklarationszeile keinen neuen Wert erhalten, und UnoArduSim erkannte 'enum'-Mitglieder nicht, wenn auf sie mit einem (legal) verwiesen wurde. `'enumname::'` Präfix.

V2.2– Jun. 2018

- 1) Das Aufrufen eines funktionsmodul mit weniger Argumenten als von seiner Definition benötigt (als dieser funktionsmodul "vorwärts definiert" war, dh als er keine frühere prototyp-Deklarationszeile hatte), verursachte eine Speicherverletzung und einen Absturz.
- 2) Seit V2.1 **Wellenformen** wurde während nicht aktualisiert **Ausführen** (nur bei **Halt** oder nach a **Schritt**) - in Ergänzung, **VariablenBereich** Die Werte wurden über einen längeren Zeitraum nicht aktualisiert **Schritt**

Operationen.

- 3) Einige minderjährige **Wellenform** Probleme beim Scrollen und Zoomen, die seit V2.0 bestanden, wurden behoben.
- 4) Schon in frühen Versionen hat der Zurücksetzen bei $t = 0$ einen PULSER oder FUNCGEN, dessen Periode seinen allerletzten Zyklus vor $t = 0$ machen würde sei nur ein teilweise Zyklus führte zu seiner **Wellenform** nach dem $t = 0$ wird von seiner wahren Position um diesen (oder den verbleibenden) Teilzyklusbetrag entweder nach rechts oder nach links verschoben.
- 5) Einige Probleme mit der Hervorhebung von Syntaxfarben in wurden behoben **Editieren/Ansehen** .
- 5) Seit V2.0 funktionierte das Klicken auf erweitern eines objekt in einem array von objekte nicht richtig.
- 6) '`delay(long)`' wurde korrigiert, um zu sein '`delay(unsigned long)`' und '`delayMicroseconds(long)`' wurde korrigiert, um zu sein '`delayMicroseconds(unsigned int)`' .
- 7) Ab V2.0 wird funktionsmodule mit angeschlossen '`attachInterrupt()`' wurden für diesen Zweck nicht als gültig funktionsmodule überprüft (dh '`void`' return und ohne Aufrufparameter).
- 8) Der Einfluss von '`noInterrupts()`' auf funktionsmodule '`micros()`', '`mills()`', '`delay()`', '`pulseInLong()`'; seine Auswirkungen auf '`Stepper::step()`' und auf '`read()`' und '`peek()`' Zeitüberschreitungen; bei allen RX-Serienempfängern und bei '`Serial`' Übertragung wird nun genau wiedergegeben.
- 9) Die in Benutzerinterruptroutinen verbrachte Zeit wird jetzt in dem von zurückgegebenen Wert berücksichtigt '`pulseIn()`', die Verzögerung von '`delayMicroseconds()`' und an der Position der angezeigten Kanten **Pin Digital Wellenformen** .
- 10) Anrufe nach **objekt-Mitglied** funktionsmodule, die Teil größerer komplexer Ausdrücke waren oder selbst in funktionsmodul-Aufrufen mit mehreren komplexen Argumenten enthalten waren, z. '`myobj.memberfunc1() + count/2`' oder '`myfunc(myobj.func1(), count/3)`', hätten falsche Werte zur ausführung-Zeit berechnet / übergeben, da die Stapelspeicherzuordnungen fehlerhaft waren.
- 11) Arrays des Zeigers variablen funktionierte ordnungsgemäß, es wurden jedoch fehlerhafte Anzeigewerte in angezeigt **VariablenBereich**.
- 12) Beim dynamischen arrays wurden einfache Typen mit angelegt '`new`', nur das erste Element hat eine (Standard-) Initialisierung auf den Wert 0 bekommen - jetzt tun es alle Elemente.
- 13) '`noTone()`', oder das Ende eines endlichen Tons setzt den pin nicht mehr zurück (er bleibt bestehen) '`OUTPUT`' und geht '`LOW`').
- 14) Kontinuierliche Rotation 'SERVO' geräte sind jetzt bei einer Pulsbreite von 1500 Mikrosekunden perfekt stationär.
- 15) Das Aufrufen von `SdFile::ls()` (SD-Kartenverzeichnisliste) funktionierte ordnungsgemäß, zeigte jedoch in Waveforms fenster einige doppelte Block-SPI-Übertragungen nicht ordnungsgemäß an.

V2.1.1- März 2018

- 1) Es wurden Inkonsistenzen in nicht englischen Gebietsschemata mit der Sprache behoben, in der gespeichert wurde '**myArduPrefs.txt**', mit angezeigten Optionsfeldern in der **Präferenzen** Dialogfeld und mit Übereinstimmung mit übersetzten Zeilen in '**myArduPrefs.txt**'.
- 2) Zuweisungen mit '`new`' Akzeptieren Sie jetzt eine ganzzahlige array-Dimension, die keine Konstante ist.
- 3) Klicken Sie in das **VariablenBereich** zu erweitern würde ein mehrdimensionales array ein überflüssiges Leerzeichen zeigen '`[]`' Klammerpaar .
- 4) Array-Elementreferenzen mit nachgestellten überflüssigen Zeichen (z. B. '`y[2]12`') wurden zur Analysieren-Zeit nicht als Fehler abgefangen (die zusätzlichen Zeichen wurden einfach ignoriert).

V2.1- März 2018

- 1) Ein fehler in den neuen Versionen V2.0.x ließ den Fenster - Heap mit jedem Update im wachsen **VariablenBereich** -- nach dem Millionen von Updates (viele Minuten im Wert von ausführung), ein Absturz könnte

die Folge sein.

2) Anrufe nach 'static' Mitglied funktionsmodule mit Doppelpunkt ' : : ' Notation fehlgeschlagen, um Analysieren im Inneren ' if() ', ' while() ', ' for() ', und ' switch() ' Klammern und Ausdrücke, die als funktionsmodul-Aufrufargumente oder array-Indizes verwendet werden.

V2.0.2 Feb. 2018

1) Ein in V2.0 eingeführtes fehler verursachte ein **Datei | Laden** Absturz, wenn ein '#include' verwiesen auf ein fehlendes oder leeres datei

2) In einem **IODEVS.TXT** datei, er 'I/O' der Name 'One-Shot' wurde anstelle des älteren 'Oneshot' erwartet; beide werden jetzt akzeptiert.

V2.0.1– Jan. 2018

3) In nicht-englischen Gebietsschemas 'en' wurde fälschlicherweise als ausgewählt in angezeigt **Präferenzen** Das Zurückkehren auf Englisch ist umständlich (erfordert eine Deaktivierung und dann eine erneute Auswahl).

4) Es war dem Benutzer möglich, einen Gerät-pin-Eingabefeldwert in einem unvollständigen Zustand (wie 'A_') und die 'DATA'-Bits eines 'SRS:V' unvollständig zu lassen.

5) Die maximale Anzahl der Analoge-Slider war auf 4 begrenzt (jetzt auf 6 korrigiert).

6) UnoArduSim besteht nicht mehr darauf ' = ' Wird in einer array-Aggregatinitialisierung angezeigt.

7) UnoArduSim hatte darauf bestanden, das Argument "inverted_logic" anzugeben ' SoftwareSerial() '.

8) Bit-Shift-Operationen erlauben nun Verschiebungen, die länger sind als die Größe des verschobenen variable.

V2.0– Dez. 2017

1) Alle funktionsmodule, die als deklariert wurden 'unsigned' hatte dennoch Werte zurückgegeben, als ob sie waren 'signed' . Dies hatte keine Auswirkung, wenn die 'return' Wert wurde einem zugewiesen 'unsigned' variable, hätte aber einen unsachgemäßen verursacht negative Interpretation, wenn es MSB == 1 hatte, und es wurde dann einem zugewiesen 'signed' variable, oder in einer Ungleichung getestet.

2) Analoge Sliders erreichten nur ein Maximum 'analogRead()' Wert von 1022, nicht die richtige 1023.

3) Ein versehentlich in V1.7.0 eingeführter fehler in der Logik zur Beschleunigung der Handhabung des SPI-Systems SCK pin verursachte SPI-Übertragungen für 'SPI_MODE1' und 'SPI_MODE3' nach dem ersten übertragenen Byte fehlschlagen (ein falsches Extra SCK-Übergang folgte jedem Byte. Außerdem wurden Aktualisierungen einer 'SPISLV'-Bearbeitungs-'DATA'-Box für übertragene Bytes verzögert.

4) Der farbige LED gerät führte 'B' (für Blau) nicht als Farboption auf (obwohl dies akzeptiert wurde).

5) Die Einstellungen für 'SPISLV' und 'I2CSLV' geräte wurden nicht im Benutzer gespeichert 'I/O' Geräte datei.

6) Kopieren 'Servo' Instanzen sind aufgrund eines Fehlers fehlgeschlagen 'Servo::Servo(Servo &toCopy)' Kopie-Konstruktor-Implementierung.

7) Außer Reichweite 'Servo.writeMicroseconds()' Werte wurden korrekt als Fehler erkannt, die angegebenen Grenzwerte im Fehlermeldungstext waren jedoch falsch.

8) Ein legaler baudrate von 115200 wurde nicht akzeptiert beim Laden von einem 'I/O' Geräte Text datei.

9) Elektrische pin konflikte, die durch einen angeschlossenen Analoger Schieberegler gerät verursacht wurden, wurden nicht immer erkannt.

10) In seltenen Fällen wird ein fehlerhafter Zeichenfolgenzeiger (ohne Abschlusszeichen für die Zeichenfolge 0) an a übergeben 'string' funktionsmodul kann UnoArduSim zum Absturz bringen.

11) Die **CodeBereich** könnte in der markieren die aktuelle Analysieren Fehlerzeile stehen **falsch** programm Modul (wenn '#include' wurde benutzt).

12) Das Laden eines 'I/O' Geräte datei mit einem gerät, der antreiben (nicht korrekt) gegen 'Uno' pin 13 austauschen würde, verursachte einen programm-Hang beim Popup der Fehlermeldung.

- 13) UnoArduSim hatte fälschlicherweise erlaubt Der Benutzer muss Nicht-Hex-Zeichen in den erweitert-TX-Puffer fenster für SPISLV und I2CSLV einfügen.
- 14) Deklarationszeileninitialisierungen fehlgeschlagen, wenn der Wert auf der rechten Seite der Wert war `'return'` Wert von einem objekt-Mitglied - funktionsmodul (wie in `'int angle = myservo1.read();'`).
- 15) `'static'` Mitglied variablen ausdrücklich haben `'ClassName::'` Präfixe wurden nicht erkannt, wenn sie am Anfang einer Zeile standen (z. B. bei einer Zuordnung zu einer Basis-'class' variable).
- 16) Berufung `'delete'` auf einen Zeiger von `'new'` wurde nur erkannt, wenn die Notation funktionsmodul runde-klammer verwendet wurde, wie in `'delete(pptr)'`.
- 17) UnoArduSim Implementierung von `'noTone()'` bestand fälschlicherweise darauf, dass ein pin-Argument angegeben wurde.
- 18) Änderungen, die die globalen 'RAM'-Bytes in einem verwendeten programm erhöhten `'String'` variablen (über **Editieren/Ansehen** oder **Datei | Laden**) könnte zu Korruption im globalen Speicherbereich von 'Uno' führen, da der Heap - Speicher gelöscht wird `'String'` objekte, das zum alten programm gehört, während der zum neuen programm gehörende Heap (falsch) verwendet wird. Dies kann unter Umständen zu einem Absturz des programm führen. Obwohl ein zweiter Laden oder Analysieren das Problem löste, wurde dieser fehler endlich behoben.
- 19) Die Rückgabewerte für `'Wire.endTransmission()'` und `'Wire.requestFrom()'` waren beide bei 0 festgefahren - diese wurden nun behoben.

Änderungen / Verbesserungen

V2.8.0- Juni 2020

- 1) Neues 'Mega2560' leiterplatte hinzugefügt **Präferenzen** Option (Leiterplatte nummer == 10). Diese Eigenschaften viele weitere 'I/O' pins, 4 weitere externe Interrupts (pins 18, 19, 20, 21), drei weitere `'HardwareSerial'` Ports (auf pins 22-27) und RAM-Speicher von 2 KByte auf 8 KByte erhöht.
- 2) Das Gerät 'SFTSER' wurde in 'ALTSER' umbenannt (da es jetzt auch mit 'Serial1', 'Serial2', und 'Serial3' verwendet werden kann).
- 3) Der 'PROGIO' gerät ermöglicht jetzt eine optionale nachfolgende Liste von 4 Master-pin-Nummern zum Definieren einer explizite Zuordnung vom 4 'Uno' Slave leiterplatte pins zum 4 Master ('Uno' oder 'Mega') leiterplatte pins.
- 4) Durch Klicken in ein gerät pin-Bearbeitungsfeld wird nun die gesamte Nummer ausgewählt, um die Eingabe zu vereinfachen, und durch Klicken in ein geräte-Nummern-Bearbeitungsfeld in Konfigurieren | 'I/O' Geräte wird dasselbe getan.
- 5) Orangefarbene Hervorhebung der relevanten Quellzeile für Popups mit Analyse- und Ausführungswarnung hinzugefügt.
- 6) Unterstützung für hinzugefügt `'digitalPinToInterrupt()'` Anrufe.
- 7) Klassen erhalten jetzt immer einen Standardkonstruktor, wenn sie keinen benutzerdefinierten Konstruktor haben (auch wenn sie keine Basisklasse oder objekt-Mitglieder haben).

V2.7.0- März 2020

- 1) Neben dem aktuellen Code-Linie (grün, wenn bereit, rot zu laufen, wenn Fehler), unterhält UnoArduSim nun, für jedes Modul, das letzte vom Benutzer angeklickt oder Stack-Navigationscode-line (mit einem dunklen olivgrünen Hintergrund hervorgehoben), Herstellung es einfacher Satz und finden temporäre haltepunkt Leitungen (eine pro Modul ist nun erlaubt, aber nur die eine in der aktuell angezeigten Modul ist in der Tat bei einer 'Run-To').
- 2) Hinzugefügt neue 'I/O' geräte (und Unterstützung 3rd-Party-Bibliothek-Code), einschließlich 'SPI' und 'I2C' **Port Expander** und 'SPI' und 'I2C' **Multiplexer LED** Steuerungen und Anzeigen (LED arrays, 4-alphanumerische Zeichen, und 4-ziffer oder 8-ziffer 7-Segment-Anzeigen).

- 3) '**Wire**' Operationen werden nicht mehr von innen Benutzern Interruptroutinen nicht zugelassen (dies unterstützt externes Interrupts von einer 'I2C' Porterweiterung).
- 4) Digital Kurven zeigen nun eine Zwischenstufe (zwischen '**HIGH**' und '**LOW**'), Wenn die pin nicht angetrieben wird.
- 5) Um Verwirrung zu vermeiden, wenn sie durch über einzelnen Schritt '**SPI.transfer()**' Anweisungen, macht UnoArduSim jetzt sicher, dass angebracht 'I/O' geräte nun ihre (Logik verzögert) letzte 'SCK' Taktflanke vor dem funktionsmodul Rückkehr empfangen.
- 6) Wenn die Auto-tab-Formatierung **Vorliebe** aktiviert ist, die Eingabe eines Schließ geschweifte klammer '}' im **Editieren/Ansehen** Jetzt bewirkt einen Sprung in die Registerkarte Einrückung Position seiner passende Öffnung geschweifte klammer '{' Partner.
- 7) EIN **Re-Format** Button wurde hinzugefügt, um **Editieren/Ansehen** (Verursachen sofortige Auto-Tab-indent Umformatierung) - diese Schaltfläche nur aktiviert, wenn das Auto-Tab-indent Preference aktiviert ist.
- 8) Eine klarere Fehlermeldung tritt nun, wenn ein Präfix Schlüsselwort (wie '**const**'. '**unsigned**', oder 'PROGMEM') folgt eine Kennung in einer Erklärung (es die Kennung vorausgehen muss).
- 9) Initialisierte globale variablen, auch wenn nie später verwendet werden, werden nun immer eine Speicheradresse zugewiesen, und wird so sichtbar erscheinen.

V2.6.0 Januar 2020

- 1) Hinzugefügt character-LCD-Anzeige mit geräte 'SPI', 'I2C' und 4-bi-Parallel-Schnittstelle. Unterstützende Bibliothek Quellcode in die neue Installation 'include_3rdParty' Ordner hinzugefügt (und kann unter Verwendung eines normalen zugegriffen werden '**#include**' Richtlinie) - Benutzer kann alternativ festlegen, die statt ihre eigene funktionsmodule zu antreiben dem LCD gerät schreiben.
- 2) **CodeBereich** Hervorhebung wurde mit separaten markieren Farben für eine bereite Code-Zeile, für eine Fehlercode-Linie und für jede andere Code-Linie verbessert.
- 3) Das **Finden** Menü und Werkzeugleiste 'func' Aktionen (iv-up und die nächsten nach unten) nicht mehr springt zum vorherigen / nächsten funktionsmodul Linie starten und stattdessen jetzt ascend (oder absteigen) der Anruf-Stapel, Hervorhebung der entsprechende Code-Linie in dem Anrufer (oder angerufenen) funktionsmodul bzw. wo die **VariablenBereich** Inhalt wird angepasst variablen für die funktionsmodul mit der aktuell markierten Code-Zeile zu zeigen.
- 4) Um Verwirrung zu vermeiden, eine **Speichern** done innen **Editieren/Ansehen** bewirkt einen sofortigen Wieder**Kompilieren**Und wenn die Speichern erfolgreich war, eine nachfolgende Verwendung Stornieren oder Ausgang wird nun nur Text zurückkehren zu diesem zuletzt gespeicherten Text.
- 5) Hinzugefügt, um eine gepulste Eingangs Schrittmotor ('PSTEPR') mit 'STEP' (Impuls), 'EN*' (enable) und 'DIR' (Richtung) Eingänge und einen Mikroschritten pro Schritt der Einstellung (1,2,4,8, oder 16) .
- 6) Sowohl 'STEPR' und 'PSTEPR' geräte haben jetzt eine 'sync' LED (GRÜN für synchronisierte oder rot, wenn deaktiviert um eine weitere Schritte Erz.)
- 7) 'PULSER' geräte haben jetzt die Wahl zwischen Mikrosekunden und Millisekunden für 'Period' und 'Pulse'.
- 8) Eingebaut-funktionsmodul Auto-Vervollständigungen nicht länger beibehalten, den Parametertyp vor dem Parameternamen ein.
- 9) Beim Umschalten zurück zu einem früheren **CodeBereich**, Seine zuvor markierte Zeile wird nun erneut hervorgehoben.
- 10) Als Hilfe einer vorübergehenden Pause-Punkt Einstellung, mit Stornieren oder Ausgang aus **Editieren/Ansehen** verlässt die markieren in der **CodeBereich** auf der Linie zuletzt durch den Cursor besucht haben **Editieren/Ansehen**.
- 11) Eine benutzerdefinierte (oder 3rd-Party) '**class**' Es wird nun nutzen '**Print**' oder '**Stream**' als seine Basisklasse. Um dies zu unterstützen, wird ein neuer Ordner 'include_Sys' hinzugefügt worden ist (in dem

UnoArduSim Installationsverzeichnis), die den Quellcode für jede Basis liefert `'class'`. In diesem Fall ruft solche Base-`'class'` funktionsmodule wird identisch Benutzer behandelt werden Code (das) in gestuften werden), statt als eingebaut funktionsmodul, die nicht in gestuft sein kann (wie beispielsweise `'Serial.print()'`).

12) Member-funktionsmodul Auto-Vervollständigungen nun auch die Parameternamen **Anstatt von** seine Art.

13) Die UnoArduSim Analysieren ermöglicht nun eine objekt Namen in einer variable Erklärung seiner optionalen vorangestellt werden (und matching) `'struct'` oder `'class'` Schlüsselwort, gefolgt von der `'struct'` oder `'class'` Name.

V2.5.0 Oktober 2019

1) Zusätzliche Unterstützung für **'TFT.h'** Bibliothek (mit Ausnahme von `'drawBitmap()'`) Und ein zugehöriges **'TFT'** hinzugefügt **'I/O'** Gerät (128 mal 160 Pixel). Beachten Sie, dass, um eine übermäßige Echtzeit-Verzögerungen bei großen zu vermeiden **'fillXXX()'** Übertragung, SA Teil des **'SPI'** Transfers **in der Mitte der Füllung** wird aus dem **'SPI'** Bus fehlen.

2) Während große datei Transfers durch **'SD'**, ein Teil des **'SPI'** Transfers in der Mitte der Folge von Bytes aus dem **'SPI'** Bus ähnlich abwesend.

3) Verringert `'Stream'`-usage Overhead-Bytes, so dass **'RAM free'** Wert mehr paßt eng Arduino Kompilierung.

4) UnoArduSim warnt nun den Benutzer, wenn ein `'class'` hat mehrere Mitglieder erklärten auf einer Deklarationszeile.

5) Mit **'File | Save As'** Jetzt setzt das aktuelle Verzeichnis, dass das Verzeichnis-in gespeichert.

6) Die beiden fehlenden `'remove()'` Mitglied funktionsmodule wurden die zugegebene `'String'` Klasse.

7) UnoArduSim jetzt Verbietet Konstruktor Basis-Anrufe in einem Konstruktor-funktionsmodul prototyp es sei denn, der volle funktionsmodul Körper Definition unmittelbar folgt (um mit dem Arduino kompilierer zu vereinbaren).

8) EdGE-Übergangszeit von digital Wellenformen wurden reduziert, um Visualisierung schneller **'SPI'** Signale bei höchsten Zoom zu unterstützen.

9) Un oArduSim erlaubt nun einige Konstrukteure deklariert werden `'private'` oder `'protected'` (Für die interne Klasse Gebrauch).

V2.4 Mai 2019

1) Alle **'I/O'**-Gerätedateien werden jetzt in sprachübersetzter Form gespeichert und zusammen mit der **Präferenzen** werden sie jetzt in UTF-8-Textcodierung gespeichert, um Übereinstimmungsfehler beim nachfolgenden Lesen zu vermeiden.

2) Neu hinzugefügt **'PROGIO'** Gerät ist ein nackter programmierbarer Slave **'Uno'** leiterplatte, der bis zu 4 pins gemeinsam mit dem **LaborBankBereich** Master **'Uno'**, - kann ein Slave **'Uno'** haben **Nein** **'I/O'** geräte für sich.

3) Sie können jetzt jeden **'I/O'** gerät löschen, indem Sie ihn bei gedrückter **'Ctrl'**-Taste anklicken.

4) Im **Editieren/Ansehen** Für Global, Built-Ins und Member variablen und funktionsmodule wurde die automatische Vervollständigung von Text hinzugefügt **Eingeben** wenn in der Liste der integrierten Funktionen derzeit eine übereinstimmende Auswahl markiert ist).

5) Im **Präferenzen** ermöglicht eine neue Auswahl das automatische Einfügen eines Zeilenende-Semikolons nach n **Eingeben** Tastendruck (wenn die aktuelle Zeile eine ausführbare Anweisung ist, die in sich abgeschlossen und vollständig zu sein scheint).

6) Drücken Sie **'Ctrl-S'** von einem **Wellenform** Mit fenster können Sie alle auf einem datei speichern (**X, Y** zeigt entlang des angezeigten Abschnitts jedes wellenform (wobei X Mikrosekunden vom äußersten linken wellenform ist) Punkt und Y ist Volt).

7) Ein 'SFTSER' 'gerät hat jetzt eine versteckte (optional) 'inverted' Wert (*gilt sowohl für TX als auch für RX*) das kann nach seinem Baudratenwert am Ende seiner Zeile in einem angehängt werden **IODevs.txt** datei.

8) Füge hinzu, die 'SPISettings' Klasse, funktionsmodule 'SPI.transfer16()' , 'SPI.transfer(byte* buf, int count)' , 'SPI.beginTransaction()' , und 'SPI.endTransaction()' , ebenso gut wie 'SPI.usingInterrupt()' und 'SPI.notUsingInterrupt()' .

9) SPI-Bibliothek funktionsmodule hinzugefügt 'SPI.detachInterrupt()' zusammen mit einer SPI-Bibliothekserweiterung 'SPI.attachInterrupt(void myISRfunc)' (anstelle der eigentlichen Bibliothek funktionsmodul 'SPI.attachInterrupt(void)' (um zu vermeiden, dass generische Produkte erkannt werden müssen) 'ISR(int vector)' Low-Level-Vectored-Interrupt-funktionsmodul-Deklarationen).

10) Das SPI-System kann jetzt im Slave-Modus verwendet werden 'SS' pin (pin 10) an 'INPUT' pin und fahren es 'LOW' nach dem 'SPI.begin()' oder durch Angabe 'SPI_SLV' als optional 'mode' Parameter in 'SPI.begin(int mode=SPI_MSTR)' (Eine weitere UnoArduSim - Erweiterung für 'SPI.h'). Empfangene Bytes können dann mit gesammelt werden 'rxbyte = SPI.transfer(tx_byte)' entweder innerhalb eines Nicht-SPI-Interrupts funktionsmodul oder innerhalb eines Benutzer-Interrupt-Dienstes funktionsmodul, der zuvor von angehängt wurde 'SPI.attachInterrupt(myISRfunc)' . Im Slave-Modus 'transfer()' wartet, bis ein Datenbyte in SPDR bereit ist (blockiert also normalerweise das Warten auf einen vollständigen Byteempfang, in einer Interruptroutine jedoch **Rückkehr** sofort, da das empfangene SPI-Byte bereits vorhanden ist). In beiden Fällen, 'tx_byte' wird in den SPDR gelegt, wird also von dem angeschlossenen Master-SPI bei seinem nächsten empfangen 'transfer()' .

11) Die Unterstützung des Sklave-Modus wurde der UnoArduSim-Implementierung von 'Wire.h' hinzugefügt. Funktionsmodul 'begin(uint8_t slave_address)' ist jetzt verfügbar, so wie sie sind 'onReceive(void*)' und 'onRequest(void*)' .

12) 'Wire.end()' und 'Wire.setClock(freq)' kann jetzt angerufen werden; letztere zum Einstellen der SCL-Frequenz mit a 'freq' Wert von entweder 100.000 (Standard-SCL-Frequenz im Standardmodus) oder 400.000 (Schnellmodus).

13) 'I2CSLV' geräte reagieren jetzt alle auf die 0x00 General-Call-Bus-Adresse, und so weiter 0x00 darf nicht mehr als eindeutige I2C-Busadresse für einen dieser Slaves gewählt werden.

14) Das modellierte ausführung verzögert grundlegende Ganzzahl- und Zuweisungsoperationen und array- und Zeigeroperationen wurden reduziert, und für jede Gleitkommaoperation werden jetzt 4 Mikrosekunden hinzugefügt.

V2.3 Dez. 2018

1) Das Tracking wurde nun auf dem aktiviert **Werkzeugleiste** 'I/O ____ S' **Schieberegler** für kontinuierliches und glattes Skalierung von 'I/O' gerät-Werten, denen der Benutzer das Suffix 'S' hinzugefügt hat.

2) Ein neuer '**LED4**' 'I/O' gerät (Reihe mit 4 LEDs an) **4 aufeinanderfolgende pin-Nummern**) wurde hinzugefügt.

3) Ein neuer '**7SEG**' 'I/O' gerät (7-Segment-LED ziffer mit hexadezimalen-Code aktiviert **4 aufeinanderfolgende pin-Nummern** und mit active-low **CS** * Eingang auswählen) wurde hinzugefügt.

4) Ein neuer '**JUMP**' 'I/O' gerät, der sich wie eine Drahtbrücke zwischen zwei 'Uno' pins verhält, wurde hinzugefügt. Dies ermöglicht eine 'OUTPUT' pin zur Verdrahtung an 'INPUT' pin (siehe gerät oben für mögliche Anwendungen dieser neuen Funktion).

5) Ein neuer '**OWISLV**' 'I/O' gerät wurde hinzugefügt, und der Drittanbieter '<OneWire.h>' Bibliothek kann jetzt mit verwendet werden '#include' damit der Benutzer programme die Schnittstelle zu einer kleinen Teilmenge des '1-Wire'-Busses geräte testen kann.

6) Das **Ausführen** Speisekarte **Zurücksetzen** Befehl ist jetzt mit dem verbunden **Zurücksetzen** Taste.

7) Für mehr Klarheit, wann **Künstliche 'loop()' Verzögerung** wird unter ausgewählt **Optionen** Menü, eine explizite 'delay(1)' Der Aufruf wird unten in der Schleife eingefügt 'main()' - Dies ist jetzt eine echte Verzögerung, die durch angehängte Benutzer-Interrupts unterbrochen werden kann 'Uno' pins 2 und 3.

- 8) Elektrik pin steht konflikt mit Open-Drain oder CS-Selected, 'I/O' geräte (z. B. I2CLV oder SPISLV) werden jetzt deklariert **Nur wenn ein echtes konflikt zum Zeitpunkt ausführung auftritt**, anstatt beim ersten Anschließen des gerät sofort einen Fehler zu verursachen.
- 9) Funktionsmodul '**pulseInLong()**' ist jetzt auf 4-8 Mikrosekunden genau, um mit Arduino übereinzustimmen (die vorherige Genauigkeit betrug 250 Mikrosekunden).
- 10) Bei der Initialisierung eines globalen variable wurden Fehler gemeldet, jetzt markieren und variable im **CodeBereich**.

V2.2 Juni 2018

- 1) Auf **Speichern** von entweder der **Präferenzen** Dialogfeld oder von **Konfigurieren | 'I/O' Geräte**, das '**myArduPrefs.txt**' datei wird nun in das Verzeichnis des aktuell geladenen programm gespeichert - jedes weitere **Datei | Laden** lädt dann automatisch die datei zusammen mit den angegebenen IODEvs datei aus demselben programm-Verzeichnis.
- 2) Funktionsmodul '**pulseInLong()**' **war** fehlt, wurde aber jetzt hinzugefügt (es stützt sich auf '**micros()**' für seine Messungen).
- 3) Wenn ein Benutzer programm eine '**#include**' von a '***.h**' datei, UnoArduSim versucht nun auch automatisch das entsprechende zu laden '***.c**' datei **ob** ein entsprechender '***.cpp**' datei wurde nicht gefunden.
- 4) Automatisches Einfügen eines close-geschweifte klammer '**}**' (Nach jedem offenen geschweifte klammer '**{**') wurde hinzugefügt zu **Präferenzen**.
- 5) Ein neuer **Optionen** Menüwahl jetzt erlaubt '**interrupts()**' aus einer Benutzer-Interrupt-Routine heraus aufzurufen - dies dient nur zu Ausbildungszwecken, da das Verschachteln von Interrupts in der Praxis vermieden werden sollte.
- 6) Type-cast von Zeigern auf eine '**int**' Der Wert wird jetzt unterstützt (es wird jedoch eine Warnmeldung angezeigt).
- 7) UnoArduSim unterstützt jetzt beschriftete programm-Leitungen (z. B. '**LabelName: count++;**' für die Benutzerfreundlichkeit (aber '**goto**' ist immer noch **nicht erlaubt**)
- 8) Ausführung-Warnungen treten jetzt bei einem Anruf auf '**tone()**' könnte aktive PWM auf pins 3 oder 11 stören, wenn '**analogWrite()**' würde mit einem Servo stören, das bereits auf dem gleichen pin aktiv ist, Wenn das Eintreffen eines seriellen Zeichens verpasst wird, weil Interrupts derzeit deaktiviert sind, und wenn Interrupts so schnell eintreffen, dass UnoArduSim einige davon verpasst.

V2.1 März 2018

- 1) Angezeigt **VariablenBereich** Die Werte werden jetzt nur alle 30 Millisekunden aktualisiert (und die Option Minimal kann diese Aktualisierungsrate noch weiter reduzieren) **VarAktualisieren** Menüoption zum Verboten der Reduzierung von Updates wurde entfernt.
- 2) Operationen, die nur auf einen Teil der Bytes eines variable-Werts abzielen (z. B. die durch Zeiger gemachten) bewirken, dass sich die Änderung dieses variable-Werts in der widerspiegelt **VariablenBereich** Anzeige.

V2.0.1 Jan. 2018

- 1) Undokumentierter Arduino funktionsmodule '**exp()**' und '**log()**' wurden nun hinzugefügt.
- 2) 'SERVO' geräte kann jetzt kontinuierlich gedreht werden (die Impulsbreite regelt also die Geschwindigkeit anstelle des Winkels).
- 3) Im **Editieren/Ansehenein** abschließender geschweifte klammer '**}**' wird jetzt automatisch hinzugefügt, wenn Sie eine Öffnung geschweifte klammer eingeben '**{**' wenn du das gewählt hast **Präferenz**.
- 4) Wenn Sie auf die Schaltfläche klicken **Editieren/Ansehen** fenster Titelleiste '**x**' Zum Beenden haben Sie nun die Möglichkeit, den Vorgang abubrechen, wenn Sie den angezeigten programm datei geändert, aber nicht gespeichert haben.

V2.0 Sept. 2017

1) Die Implementierung wurde auf QtCreator portiert, sodass die grafische Benutzeroberfläche einige geringfügige visuelle Unterschiede aufweist, jedoch abgesehen von einigen Verbesserungen keine funktionalen Unterschiede:

- a) Die Statuszeile am unteren Rand des Main fenster und in der **Editieren/Ansehen** Das Dialogfeld wurde verbessert und eine markieren-Farbcodierung hinzugefügt.
- b) Der vertikale Raum, der zwischen dem **CodeBereich** und **VariablenBereich** ist jetzt über eine ziehbare (aber nicht sichtbare) Trennleiste an ihrem gemeinsamen Rand einstellbar.
- c) 'I/O' gerät-Eingabefeldwerte werden jetzt erst validiert, wenn der Benutzer den Mauszeiger außerhalb des gerät bewegt hat. Auf diese Weise werden umständliche automatische Änderungen zur Durchsetzung zulässiger Werte während der Eingabe durch den Benutzer vermieden.

2) UnoArduSim unterstützt jetzt mehrere Sprachen über **Konfigurieren | Präferenzen**. Zusätzlich zur Sprache für das Gebietsschema des Benutzers kann immer Englisch ausgewählt werden (sofern eine benutzerdefinierte *.qm-Übersetzung datei für diese Sprache im UnoArduSim 'translations'-Ordner vorhanden ist).

3) Der Sound wurde jetzt geändert, um die Qt-Audio-API zu verwenden. Dies erforderte unter bestimmten Umständen eine Stummschaltung. Vermeiden Sie störende Geräusche und Klicks. Während des längeren Betriebssystemfensters treten Betriebsverzögerungen auf, die durch normale Mausklicks verursacht werden. Weitere Informationen finden Sie im Abschnitt -Sounds darauf.

4) Aus Gründen der Benutzerfreundlichkeit werden Leerzeichen jetzt verwendet, um einen 0-Wert in den gerät-Count-Bearbeitungsfeldern in darzustellen **Konfigurieren | 'I/O' Geräte** (so können Sie jetzt die Leertaste verwenden, um geräte zu entfernen).

5) Das Unskalierte Das (U) -Qualifikationsmerkmal ist jetzt für 'PULSER', 'FUNCGEN' und '1SHOT' geräte optional (dies ist die angenommene Standardeinstellung).

6) UnoArduSim jetzt erlaubt (zusätzlich zu wörtlichen numerischen Werten) '**const**' ganzzahliges variablen und 'enum'