

UnoArduSimV2.8.1 Aide Complète

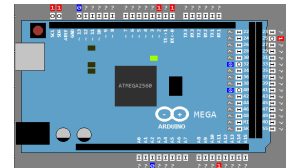


Table des Matières

[Vue d'Ensemble](#)

[Volet de Code, Préférences et Modifier/Examiner](#)

[Volet de Code](#)

[Préférences](#)

[Modifier/Examiner](#)

[Volet de Variables et Modifier/Surveiller Variable](#)

[Volet du Banc de Laboratoire](#)

[le 'Uno' ou 'Mega'](#)

['I/O' Dispositifs](#)

['Serial' Moniteur \('SERIAL'\)](#)

[Alterner Série \('ALTSER'\)](#)

[Lecteur de Disque SD \('SD_DRV'\)](#)

[Écran TFT \('TFT'\)](#)

[SPI Esclave Configurable \('SPISLV'\)](#)

[I2C Esclave TwoWire \('I2CSLV'\)](#)

[Texte LCD I2C \('LCDI2C'\)](#)

[Texte LCD SPI \('LCSPI'\)](#)

[Texte LCD D4 \('LCD_D4'\)](#)

[Multiplexeur I2C DEL \('MUXI2C'\)](#)

[Multiplexeur SPI DEL \('MUXSPI'\)](#)

[Port d'Expansion SPI \('EXPSPi'\)](#)

[Port d'Expansion I2C \('EXPI2C'\)](#)

['1-Wire' Esclave \('OWIISLV'\)](#)

[Programmable 'I/O' Dispositif \('PROGIO'\)](#)

[Registre de Décalage Esclave \('SRSLV'\)](#)

[Générateur Un-Tir \('1SHOT'\)](#)

[Pulseur Binaire \('PULSER'\)](#)

[Analogique Générateur de Fonction \('FUNCGEN'\)](#)

[Moteur Pas à Pas \('STEPR'\)](#)

[Moteur Pas à Pas Pulsé \('PSTEPR'\)](#)

[DC Moteur \('MOTOR'\)](#)

[ServoMoteur \('SERVO'\)](#)

[Haut-parleur Piézo \('PIEZO'\)](#)

[Résistance de glissière \('R=1K'\)](#)

[Bouton Poussoir \('PUSH'\)](#)

[DEL coloré \('LED'\)](#)

[4-DEL Rangée \('LED4'\)](#)

[7 segments DEL Chiffre \('7SEG'\)](#)

[Curseur Analogique](#)

[Fil de Connexion du Pins \('JUMP'\)](#)

[Les Menus](#)

[Fichier:](#)

[Chargez INO ou PDE Prog \(ctrl-L\)](#)

[Modifier/Examiner \(ctrl-E\)](#)

[Enregistrer](#)

[Enregistrer Sous](#)

[Suivant \('#include'\)](#)

[précédent](#)

[Quitter](#)

[Trouver:](#)

[Monter Pile d'Appels](#)

[Descendre Pile d'Appels](#)

[Définir le texte Rechercher \(ctrl-F\)](#)

[Trouver Texte suivant](#)

[Trouver Texte précédent](#)

[Exécuter:](#)

[Un Pas Dans \(F4\)](#)
[Un Pas Franchir \(F5\)](#)
[UnPas Sortir \(F6\)](#)
[Exécuter Vers \(F7\)](#)
[Exécuter Jusqu'à \(F8\)](#)
[Exécuter \(F9\)](#)
[Arrêtez \(F10\)](#)
[Réinitialiser](#)
[Animer Exécution](#)
[Ralenti](#)

[Options:](#)

[Un Pas Franchir Structors/Les opérateurs](#)
[Registre-Allocation](#)
[Erreur sur non initialisé](#)
[Ajoutée 'loop\(\)' Retard](#)
[Autoriser les interruptions imbriquées](#)

[Configurer:](#)

['I/O' Dispositifs](#)
[Préférences](#)

[VarRafraîchir:](#)

[Autoriser Auto \(-\) Contract](#)
[Minimal](#)
[Surlinger Changements](#)

[Fenêtres:](#)

['Serial' Moniteur](#)
[Tout restaurer](#)
[Pin Formes d'Ondes Numérique](#)
[Pin Forme d'Onde Analogique](#)

[Aide:](#)

[Quick Aide Fichier](#)
[Plein Aide Fichier](#)
[Bogue Corrections](#)
[Changement / Améliorations](#)
[Sur](#)

['Uno' ou 'Mega' Circuit imprimé et 'I/O' Dispositifs](#)

[Timing](#)

['I/O' Dispositif Timing](#)

[Des sons](#)

[Limitations et éléments non pris en charge](#)

[Fichiers inclus](#)

[Allocation dynamique de mémoire et RAM](#)

[Allocations de mémoire 'Flash'](#)

['String' Variables](#)

[Bibliothèques Arduino](#)

[Pointeurs](#)

['class' et 'struct' Objets](#)

[Échelle](#)

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

[Directives Acompilateur](#)

[Éléments de langue Arduino](#)

[C / C ++ - éléments de langage](#)

[Fonction Modèles](#)

[Emulation en temps réel](#)

[Notes de version](#)

[Bogue Corrections](#)

[V2.8.1- Juin 2020](#)

[V2.8.0- Juin 2020](#)

[V2.7.0- Mars 2020](#)

[V2.6.0- Jan 2020](#)

[V2.5.0- octobre 2019](#)

[V2.4- mai 2019](#)

[V2.3- décembre 2018](#)

[V2.2– juin. 2018](#)

[V2.1.1– mars 2018](#)

[V2.1– mars 2018](#)

[V2.0.2 février 2018](#)

[V2.0.1– janvier 2018](#)

[V2.0– Déc. 2017](#)

[Changements / Améliorations](#)

[V2.8.0- Juin 2020](#)

[V2.7.0 Mars 2020](#)

[V2.6.0 janvier 2020](#)

[V2.5.0 octobre 2019](#)

[V2.4 mai 2019](#)

[V2.3 déc. 2018](#)

[V2.2 juin 2018](#)

[V2.1 mars 2018](#)

[V2.0.1 janvier 2018](#)

[V2.0 septembre 2017](#)

Vue d'Ensemble

UnoArduSim est un logiciel gratuit **temps réel** (voir pour Timing **restrictions**) un outil de simulation que j'ai développé pour les étudiants et les passionnés d'Arduino. Il est conçu pour vous permettre d'expérimenter et de déboguer facilement Arduino programmes. **sans avoir besoin de matériel réel** . Il est destiné à la **Arduino 'Uno' ou 'Mega'** circuit imprimé et vous permet de choisir parmi un ensemble de 'I/O' virtuels dispositifs et de configurer et connecter ces dispositifs à votre 'Uno' ou 'Mega' virtuel dans le **Volet du Banc de Laboratoire** . - vous n'avez pas à vous soucier des erreurs de câblage, des connexions brisées ou desserrées, ou du dispositifs défectueux gênant le développement et les tests de votre programme.

UnoArduSim fournit des messages d'erreur simples pour toutes les erreurs analyser ou exécution rencontrées et permet le débogage avec **Réinitialiser** , **Exécuter** , **Exécuter Vers**, **Exécuter Jusqu'à** , **Arrêtez** et flexible **Un Pas** opérations dans le **Volet de Code** , avec une vue simultanée de tous les variables, tableaux et objets locaux et actifs dans la **Volet de Variables** . La vérification des limites tableau à la Exécuter est fournie et le dépassement de la mémoire RAM ATmega sera détecté (et la ligne de coupable programme mise en surbrillance!). Tous les conflit avec électriques connectés au 'I/O' dispositifs sont signalés et signalés lorsqu'ils se produisent.

Quand un INO ou PDE programme fichier est ouvert, il est chargé dans le programme **Volet de Code** . Le programme reçoit alors un Analyser, pour le transformer en un exécutable sous forme de jeton qui est alors prêt pour **exécution simulé** (contrairement à Arduino.exe, un exécutable autonome binaire est *ne pas* Toute erreur analyser est détectée et signalée en mettant en surbrillance la ligne qui a échoué en analyser et en signalant l'erreur sur la ligne. **Barre d'état** tout en bas de l'application UnoArduSim fenêtre. Un **Modifier/Examiner** fenêtre peut être ouvert pour vous permettre de voir et d'éditer une version surlignée en syntaxe de votre utilisateur programme. Les erreurs pendant la simulation de exécution (telle qu'une débit en bauds mal appariée) sont signalées dans la barre d'état et via une boîte de message contextuelle.

UnoArduSim V2.6 est une implémentation pratiquement complète de la **Langage de programmation Arduino V1.6.6 comme documenté à la arduino.cc** . Page Web Référence de langue et avec les ajouts indiqués dans les Notes de version de la page Téléchargement. Bien que UnoArduSim ne prenne pas en charge l'implémentation C ++ complète du GNU Acompilateur sous-jacent Arduino.exe, il est probable que seuls les programmeurs les plus avancés trouveront qu'un élément C / C ++ qu'ils souhaitent utiliser est manquant (et bien sûr, il existe toujours des fonctions simples). solutions de codage pour ces fonctionnalités manquantes). En général, je n'ai pris en charge que ce que je considère être les fonctionnalités C / C ++ les plus utiles pour les amateurs et les étudiants Arduino - par exemple, **'enum'** et **'#define'** sont pris en charge, mais les pointeurs fonction ne le sont pas. Même si objets défini par l'utilisateur (**'class'** et **'struct'**) et (la plupart) les surcharges d'opérateurs sont supportées, *l'héritage multiple n'est pas* .

Parce que UnoArduSim est un simulateur de langage de haut niveau, **seules les instructions C / C ++ sont prises en charge** , *les déclarations en langage assembleur ne sont pas* . De même, comme il ne s'agit pas d'une simulation de machine de bas niveau, **Les registres ATmega328 ne sont pas accessibles à votre programme** pour la lecture ou l'écriture, bien que l'allocation de registre, le passage et le retour soient émulés **Options**).

A partir de V2.6, UnoArduSim a intégré support automatique pour un sous-ensemble limité de l'Arduino a fourni des bibliothèques, qui sont: **'Stepper.h'** , **LV1** , **'SoftwareSerial.h'** , **'SPI.h'** , **FSM4** , **'OneWire.h'** , **'SD.h'** , **'TFT.h'** et **'EEPROM.h'** (version 2). V2.6 introduit un mécanisme pour 3^e partie de support de bibliothèque par l'intermédiaire d'archives prévu dans la **'include_3rdParty'** dossier qui se trouvent dans le répertoire d'installation UnoArduSim. Pour toute **'#include'** d'autres (par exemple créés par l'utilisateur) bibliothèques. Pour toute **'#include'** des bibliothèques créées par l'utilisateur, UnoArduSim **ne pas** recherchez la structure de répertoire d'installation Arduino habituelle pour localiser la bibliothèque; à la place vous **avoir besoin** copier l'en-tête (".h") correspondant et la source (".cpp") fichier dans le même répertoire que le programme fichier sur lequel vous travaillez (bien entendu, le contenu de tout **'#include'** fichier doit être parfaitement compréhensible pour l'UnoArduSim analyseur).

J'ai développé UnoArduSimV2.0 dans QtCreator avec une prise en charge multilingue et il n'est actuellement disponible que pour Fenêtres. TM. Le portage sur Linux ou MacOS est un projet d'avenir! UnoArduSim est né des simulateurs que j'avais développés au fil des années pour les cours que j'enseignais à l'Université Queen's. Les tests ont été assez nombreux, mais il reste forcément quelques bogues qui s'y cachent. Si vous souhaitez signaler un bogue, merci de le décrire (brièvement) dans un email à unoArduSim@gmail.com et **veillez à attacher votre code source Arduino programme induisant le bogue** je peux donc répliquer le bogue et le réparer. Je ne répondrai pas aux rapports individuels bogue et je ne garantis pas la chronologie des corrections dans une version ultérieure (rappelez-vous qu'il existe presque toujours des solutions de contournement!).

À votre santé,

Stan Simmons, PhD, P.Eng.
Professeur associé (retraité)
Département de génie électrique et informatique
Université Queen's
Kingston, Ontario, Canada







Volet de Code, Préférences et Modifier/Examiner


(De côté: Les exemples de fenêtres présentés ci-dessous sont tous sous un Fenêtres-OS choisi par l'utilisateur. thème de couleur qui a une couleur de fond bleu foncé fenêtre).

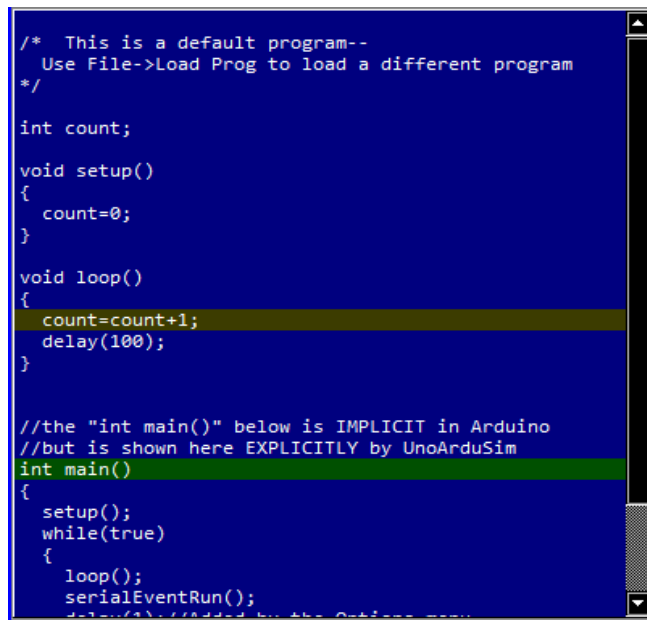
Volet de Code

le **Volet de Code** affiche votre utilisateur programme et met en surbrillance son exécution. Après qu'un programme chargé ait un Analyser réussi, la première ligne de '**main()**' est mise en surbrillance et le programme est prêt pour l'exécution. Notez que '**main()**' est implicitement ajouté par Arduino (et par UnoArduSim) et vous faites **ne pas** incluez-le en tant qu'utilisateur programme fichier. Exécution est sous le contrôle du menu **Exécuter**, et ses associés **Barre d'outils** boutons et des raccourcis clavier fonction.

Après avoir quitté exécuté par une des instructions (ou plus) (vous pouvez utiliser **Outil-Bar** boutons , , , ou ), La ligne pins qui sera exécuté est ensuite alors en vert - la ligne verte en surbrillance est toujours la ligne suivante **prêt à être exécuté**.

De même, lorsqu'un programme en cours d'exécution frappe un **Exécuter Vers** point d'arrêt, exécution est arrêté et la ligne point d'arrêt est mise en surbrillance (et est alors prête pour exécution).

Si pins exécuté est actuellement interrompue, et vous cliquez dans la **Volet de Code** formes d'Ondes, la ligne que vous venez de cliquer est mise en surbrillance dans olive foncé (comme le montre la photo) - la prochaine à être-exécute ligne toujours des séjours surlignés en vert (au V2.7). Mais vous pouvez causer exécution *progresser jusqu'à* la ligne que vous venez de surligner en cliquant ensuite sur le **Exécuter Vers**  **Barre d'outils** bouton. Cette fonctionnalité vous permet d'atteindre rapidement et facilement des lignes spécifiques dans un programme, de sorte que vous puissiez ensuite passer pas à pas ligne par ligne sur une partie de programme présentant un intérêt.





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






int count;

void setup()
{
  count=0;
}

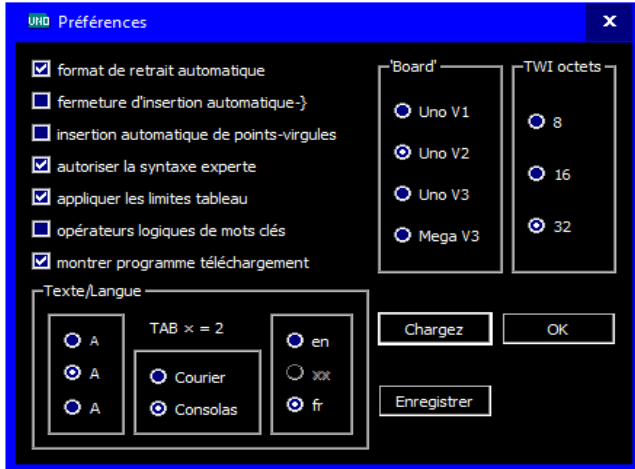
void loop()
{
  count=count+1;
  delay(100);
}

//the "int main()" below is IMPLICIT in Arduino
//but is shown here EXPLICITLY by UnoArduSim
int main()
{
  setup();
  while(true)
  {
    loop();
    serialEventRun();
    delay(1); //added by the Settings menu
```

Si votre programme chargé a des '**#include**' fichiers, vous pouvez vous déplacer entre eux en utilisant **Fichier | précédent** et **Fichier | Suivant** (avec **Barre d'outils** boutons  et ).

L'actions du menu **Trouver** vous permettent de **SOIT** trouver dans le texte **Volet de Code** ou **Volet de Variables** (**Outil-Bar** boutons  et , ou les raccourcis clavier **flèche vers le haut** et **flèche vers le bas**) **après la première utilisation** **Trouver | Définir le texte Rechercher** ou **Outil-Bar** , **OU BIEN** à **naviguer dans le call-pile** dans le **Volet de Code** (**Outil-Bar** boutons  et , ou les raccourcis clavier **flèche vers le haut** et **flèche vers le bas**). Clés **PgDn** et **PgPréc** sauter à la sélection suivante / précédente fonction ..

Préférences

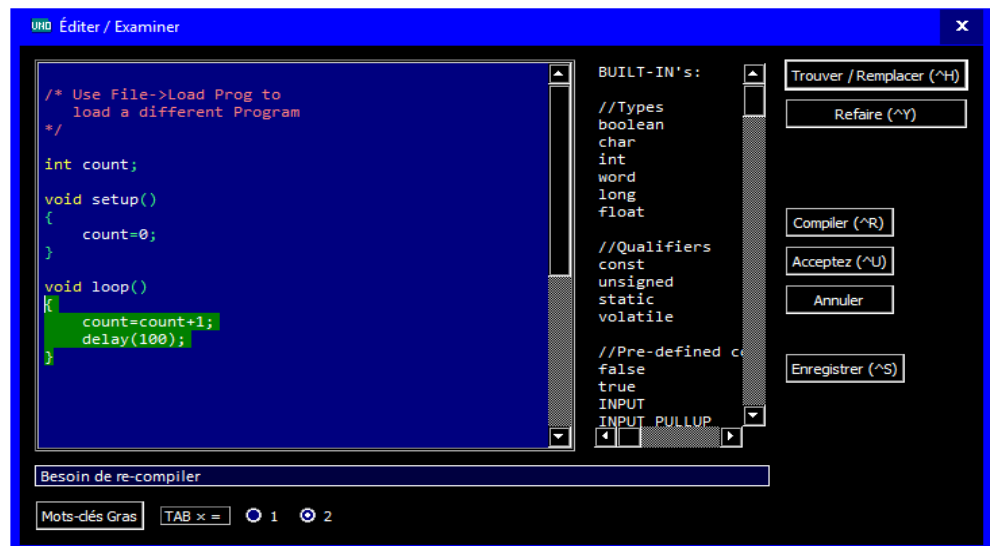


n'apparaissent que si une traduction ".qm" fichier existe dans le dossier des traductions (à l'intérieur de le répertoire de base UnoArduSim.exe).

Modifier/Examiner

En double-cliquant sur une ligne du répertoire **Volet de Code** (ou en utilisant le menu **Fichier**), un **Modifier/Examiner** fenêtre est ouvert pour permettre la modification de votre programme fichier, avec le **ligne actuellement sélectionnée** dans le **Volet de Code** est mis en évidence.

Ce fenêtre a une capacité d'édition complète avec mise en surbrillance de la syntaxe dynamique (différentes couleurs du surligner sont utilisées pour les mots-clés, commentaires, etc. C ++). La mise en évidence de la syntaxe gras et le formatage automatique au niveau de retrait sont facultatifs (en supposant que vous avez sélectionné **Configurer | Préférences**). Vous pouvez également facilement sélectionner les appels intégré fonction (ou intégré '**#define**' constantes) à ajouter à votre programme à partir de la list box fournie - Il suffit de double-cliquer sur l'élément de la liste déroulante souhaité pour l'ajouter à votre programme à la position actuelle du caret (fonction-call variable **les types** ne sont là que pour votre information et sont supprimés pour laisser des espaces fictifs lorsqu'ils sont ajoutés à votre programme).



Le fenêtre a **Trouver** (utilisation **ctrl-F**) et **Trouver / Remplacer** capacité (utilisation **ctrl-H**). le **Modifier/Examiner** fenêtre a **Reprendre** (**ctrl-Z**), et **Refaire** (**ctrl-Y**) boutons (qui apparaissent automatiquement).

De se défaire **tous les changements** depuis que vous avez ouvert le programme pour la première fois, cliquez sur le bouton **Annuler** bouton. Accepter le état actuel, cliquez sur le **Accepter** et le programme reçoit automatiquement un autre Analyser (et est téléchargé sur le 'Uno' ou 'Mega' si aucune erreur n'est détectée) et le nouvel état apparaît dans le fenêtre principal UnoArduSim **Barre d'état** .

UNE **Compiler** (**ctrl-r**) bouton (plus un associé **Statut Analyser** comme indiqué dans l'image ci-dessus) a été ajouté pour permettre de tester les modifications sans avoir à fermer le fenêtre au préalable. UNE **Enregistrer** (**ctrl-s**) a également été ajouté en tant que raccourci (équivalent à un **Accepter** plus un séparé plus tard **Enregistrer** de la fenêtre principale).

Utilisez ALT-flèche-droite de demander des choix d'auto-complétion pour intégré variables **mondiale**, et pour **nombre variables et fonctions**.

Sur l'un ou l'autre **Annuler** ou **Accepter** sans modifications apportées, le **Volet de Code** la ligne actuelle change pour devenir le **dernière position de curseur Modifier/Examiner**, et vous pouvez utiliser cette fonctionnalité pour sauter le **Volet de Code** à une ligne spécifique (éventuellement pour préparer une **Exécuter Vers**), Vous pouvez aussi utiliser **Ctrl-PgDn** et **ctrl-pgup** pour passer à la prochaine (ou précédente) rupture de ligne vide dans votre programme - cela est utile pour naviguer rapidement vers le haut ou le bas des emplacements significatifs (comme les lignes vides entre fonctions). Vous pouvez aussi utiliser **ctrl-Home** et **ctrl-End** pour aller au début et à la fin du programme, respectivement.




'**Tab**' niveau de mise en forme automatique de retrait se fait lorsque le formes d'Ondes ouvre, si cette option a été mis sous **Configurer | Préférences**. Vous pouvez refaire que le formatage à tout moment en cliquant sur le **Reformater** bouton (il est activé uniquement si vous avez précédemment sélectionné la **indentation automatique préférence**). Vous pouvez également ajouter ou vous-même des onglets de suppression à un groupe de lignes consécutives pré-sélectionnées à l'aide du clavier **flèche droite** ou **Flèche gauche** clés - mais **indentation automatique préférence doit être désactivée** pour éviter de perdre vos propres niveaux de l'onglet personnalisé.

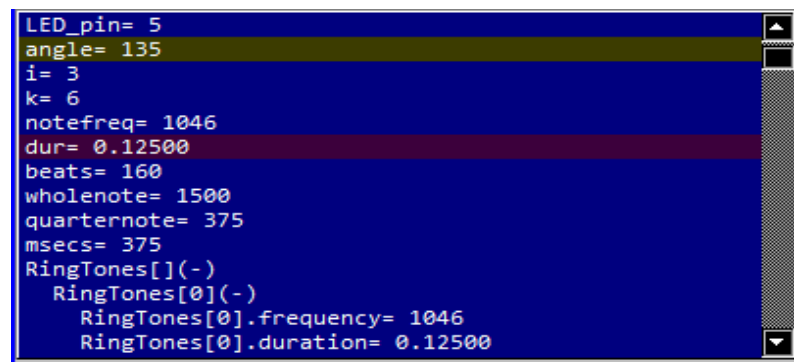
Quand **Point-virgule automatique** est cochée, en appuyant sur **Entrer** mettre fin à une ligne insère automatiquement le point-virgule de fin de ligne.

Et pour vous aider à mieux suivre vos contextes et accolades, cliquez sur un bouton ' { ' ou ' } ' accolade **met en surbrillance tout le texte entre ce accolade et son partenaire correspondant**.

Volet de Variables et Modifier/Surveiller Variable

le **Volet de Variables** est situé juste en dessous de la **Volet de Code**. Il affiche les valeurs actuelles pour chaque utilisateur variable / tableau / objet local et global (in-échelle) actif dans le programme chargé. Au fur et à mesure que votre programme exécution se déplace entre fonctions, **le contenu change pour ne refléter que les variables locaux accessibles aux fonction / échelle actuels, ainsi que tous les globaux déclarés par l'utilisateur**. Tout variables déclaré comme '**const**' ou comme '**PROGMEM**' (attribué à 'Flash' mémoire) ont des valeurs qui ne peuvent pas changer, et pour économiser de *Pas affichée*. '**Servo**' et '**SoftwareSerial**' Les instances objet ne contiennent aucune valeur utile et ne sont donc pas affichées.

Vous pouvez **trouver** spécifié **texte** avec ses commandes de recherche de texte (avec **Barre d'outils** boutons  et , ou des raccourcis clavier **flèche vers le haut** et **flèche vers le bas**), après la première utilisation **Trouver | Set Rechercher** texte ou .

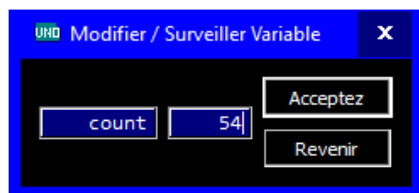


Tableaux et objets sont montrés soit **un-agrandie** ou **agrandie** format, avec soit un tirant plus ' (+) ' ou moins ' (-) ' signe, respectivement. Le symbole pour un tableau **x** montre que '**x[]**'. Pour que le agrandir affiche tous les éléments du tableau, il suffit de cliquer une fois sur '**x[] (+)**' dans le **Volet de Variables**. Pour revenir à une vue non-agrandie, cliquez sur le bouton '**x[] (-)**'. La valeur par défaut un-agrandie pour un objet '**p1**' montre que '**p1 (+)**'. Pour agrandir il montrer à tous les membres de cette '**class**' ou '**struct**' par exemple, un simple clic sur '**p1 (+)**' dans le **Volet de Variables**. Pour revenir à une vue non-agrandie, cliquez une fois sur '**p1 (-)**'

Si vous *un seul clic sur une ligne de hexadécimal en olive foncé* (Il peut être simple taille de police, ou l'agrégat **lv1** ou ' (-) ' une ligne de tableau ou surlinger, ou un élément simple ou tableau surlinger-member), faire alors **Exécuter Jusqu'à** provoquera exécuté pour reprendre et geler à la prochaine *accès en écriture* nulle part à l'intérieur de cet ensemble sélectionné, ou à celui sélectionné seul emplacement taille de police.

Lors de l'utilisation **Un Pas** ou **Exécuter**, les mises à jour des valeurs variable affichées sont effectuées en fonction des paramètres utilisateur définis dans le menu **VarRafraîchir** - cela permet une gamme complète de comportements allant des mises à jour périodiques minimales aux mises à jour immédiates complètes. Des mises à jour réduites ou minimales sont utiles pour réduire la charge du processeur et peuvent être nécessaires pour empêcher le exécution de prendre du retard en temps réel par rapport à ce qui serait autrement excessif. **Volet de Variables** fenêtre charges de mise à jour. Quand **Animer Exécution** est en vigueur ou si le **Surlinger Changements** l'option de menu est sélectionnée, la valeur d'un variable est modifiée pendant **Exécuter** sa valeur affichée sera mise à jour **immédiatement**, et il devient en surbrillance - cela entraînera la **Volet de Variables** faire défiler (si nécessaire) jusqu'à la ligne contenant variable et exécution ne seront plus en temps réel !.

Quand exécution gèle après **Un Pas**, **Exécuter Vers**, **Exécuter Jusqu'à**, ou **Exécuter** -puis- **Arrêtez**, la **Volet de Variables** met en évidence le variable correspondant à la **adresse (s) qui ont été modifiés** (le cas échéant) par le **toute dernière instruction** pendant que exécution (y compris par les initialisations de déclaration variable). Si cette instruction **complètement** rempli un **objet ou tableau**, la **parent (+) ou (-) ligne** pour cet agrégat devient en surbrillance. Si, à la place, l'instruction modifiait un emplacement qui est actuellement visible, alors il devient en surbrillance. Mais si le ou les lieux modifiés se cachent actuellement à l'intérieur un-agrandie tableau ou objet, cet agrégat **ligne parent** obtient un **police italique** comme un repère visuel que quelque chose à l'intérieur a été écrit - en cliquant sur agrandir, il va alors causer sa **dernier** élément ou membre modifié à mettre en surbrillance.



le **Modifier/Surveiller Variable** fenêtre vous donne **la capacité de suivre n'importe quelle valeur variable au cours de exécution**, ou pour **change sa valeur au milieu de (arrêtée) programme exécution** (afin que vous puissiez tester quel serait l'effet de continuer avec cette nouvelle valeur). **Arrêtez** exécution d'abord, ensuite **double-clic gauche** sur le variable dont vous souhaitez suivre ou modifier la valeur. Pour surveiller simplement la valeur au cours de programme exécution, **laisser la boîte de dialogue ouverte** et puis

l'un des **Exécuter** ou **Un Pas** commandes - sa valeur sera mise à jour dans **Modifier/Surveiller Variable** selon les mêmes règles qui régissent les mises à jour dans le **Volet de Variables**. **Pour changer la valeur de variable**, renseignez la valeur de la zone de saisie et **Accepter**. Continuer exécution (en utilisant l'un des modèles **Un Pas** ou **Exécuter** commandes) pour utiliser cette nouvelle valeur à partir de ce moment (ou vous pouvez **Revenir** à la valeur précédente).

Sur programme Chargez ou Réinitialiser notez que tous *La valeur non initialisée variables est réinitialisée à la valeur 0 et tous les pointeurs non initialisés variables sont réinitialisés à 0x0000.*

Volet du Banc de Laboratoire

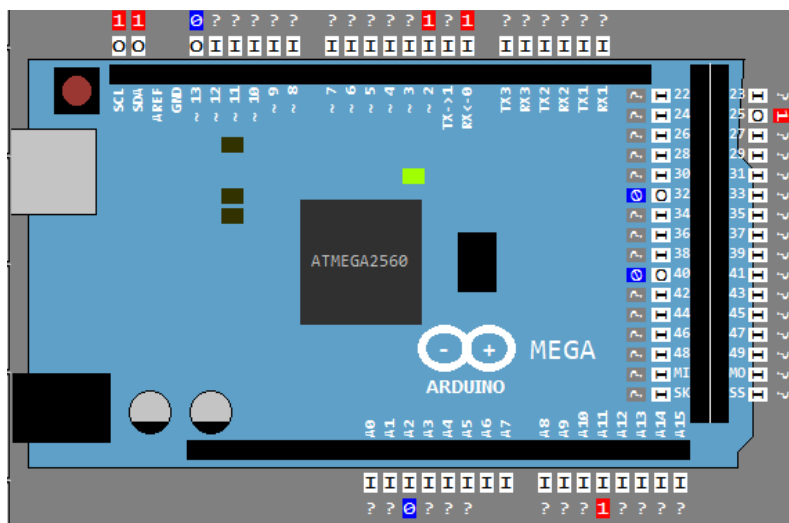
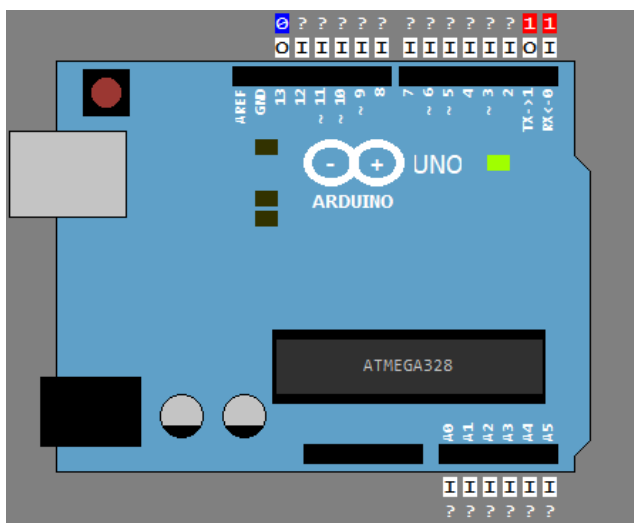
Le Volet du Banc de Laboratoire affiche un KL71 'Uno' ou 'Mega' de 5 volts, entouré d'un ensemble de K62LK 'I/O' que vous pouvez sélectionner / personnaliser et se connecter au pins souhaité.

le 'Uno' ou 'Mega'

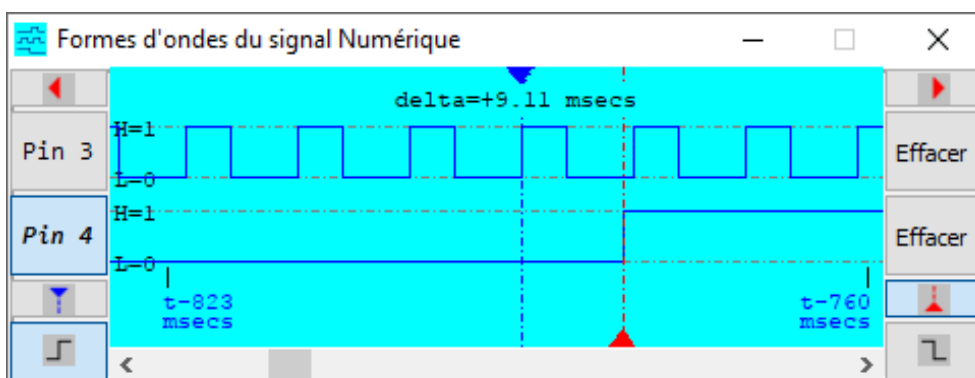
Cette est une représentation du 'Uno' ou 'Mega' ou du 'Mega' circuit imprimé et de ses LED embarquées. Lorsque vous chargez un nouveau programme dans UnoArduSim, s'il analyse correctement, il subit un "téléchargement simulé" vers le circuit imprimé qui imite la façon dont un réel Le circuit imprimé se comporte - vous verrez clignoter les séries RX et TX DEL (ainsi que l'activité sur les pins 1 et 0 qui sont *câblé pour la communication série avec un ordinateur hôte*). Ceci est immédiatement suivi par un flash pin 13 DEL qui signifie réinitialisation circuit imprimé et (et arrêt automatique UnoArduSim à) le début de votre programme exécution chargé. Vous pouvez éviter cet affichage et le décalage de chargement associé en désélectionnant **Afficher Téléchargement** de **Configurer | Préférences**.

Le fenêtre vous permet de visualiser les niveaux logiques du numérique sur les 20 'Uno' ou 'Mega' pins ou les 70 'Mega' pins ('1' sur rouge pour 'HIGH', '0' sur bleu pour 'LOW', et '?' sur gris pour une tension indéterminée indéfinie), et directions programmé ('I' pour 'INPUT', ou 'O' pour 'OUTPUT'). Pour pins qui sont pulsés en utilisant PWM via 'analogWrite()', ou par 'tone()', ou par 'Servo.write()', la couleur devient violette et le symbole affiché devient '^'.









Notez que **Numérique Les pins 0 et 1 sont câblés via des résistances de 1 kOhm sur la puce USB pour permettre communication série avec un ordinateur hôte.**





Clic gauche sur tout 'Uno' ou 'Mega' pin ouvrira une **Pin Formes d'Ondes Numérique** fenêtre qui affiche le passé **une seconde** de **Activité de niveau numérique** sur ce pin. Vous pouvez cliquer sur d'autres pins pour les ajouter à l'affichage du Pin Formes d'Ondes Numérique (jusqu'à 4 formes d'onde à la fois).



Cliquez pour afficher la page vers la gauche ou la droite, ou utilisez les touches Home, PgUp, PgDn, End.

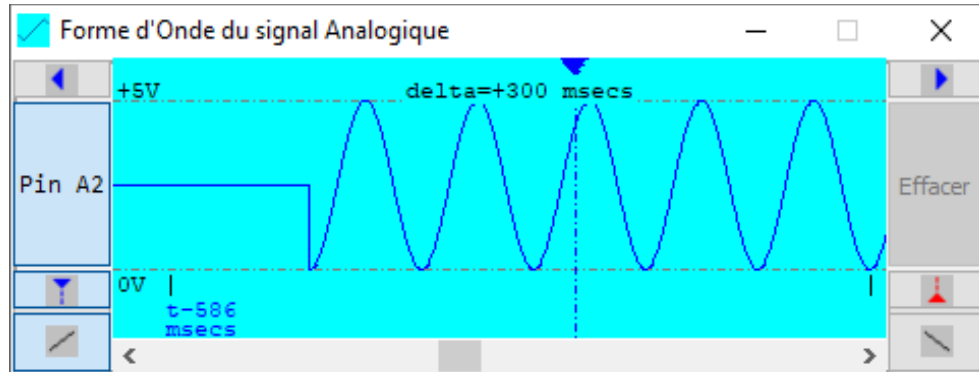
L'une des formes d'onde affichées sera la **actif pin** forme d'Onde , indiqué par le bouton "Pin" affiché comme étant enfoncé (comme dans la capture d'écran Pin Formes d'Ondes Numérique ci-dessus). Vous pouvez sélectionner un forme d'Onde en cliquant sur son bouton numéroté Pin, puis sélectionner la polarité du bord concerné en cliquant sur le bouton de sélection de polarité de front montant / descendant approprié.  , ou  , ou en utilisant les touches de raccourci **flèche vers le haut** et **flèche vers le bas** . Vous pouvez alors **saut** le curseur actif (les lignes du curseur bleu ou rouge avec leur temps delta indiqué) en arrière ou en avant jusqu'au bord numérique à polarité choisie **de ce pin actif** forme d'Onde en utilisant les boutons du curseur ( ,  ou  , ) (en fonction de quel curseur était activé plus tôt avec  ou ), ou utilisez simplement les touches du clavier ← et → .

Pour activer un curseur, cliquez sur son bouton d'activation coloré ( ou  montré ci-dessus) - *cela permet également de faire défiler la vue jusqu'à l'emplacement actuel de ce curseur* . Alternativement, vous pouvez alterner rapidement l'activation entre les curseurs (avec leurs vues respectives centrées) en utilisant le raccourci '**Tab**' clé.







Vous pouvez **saut** le curseur actuellement activé par **clic gauche n'importe où** dans la région d'affichage forme d'Onde à l'écran. Vous pouvez également sélectionner la ligne du curseur rouge ou bleu en cliquant dessus (pour l'activer), puis *faites-le glisser vers un nouvel emplacement* et relâcher. Lorsqu'un curseur souhaité est actuellement quelque part en dehors de l'écran, vous pouvez **clic droit n'importe où** dans la vue pour passer à ce nouvel emplacement à l'écran. Si les deux curseurs sont déjà à l'écran, un clic droit alterne simplement entre le curseur activé.

Pour ZOOM IN et ZOOM OUT (le zoom est toujours centré sur le curseur ACTIF), utilisez la molette de la souris ou les raccourcis clavier CTRL-flèche vers le haut et CTRL-flèche vers le bas.

Faire à la place un **clic-droit sur n'importe quel 'Uno' ou 'Mega' pin** ouvre un **Pin Forme d'Onde Analogique** fenêtre qui affiche le **passé une seconde valeur de Activité de niveau analogique** sur ce pin. Contrairement au Pin Formes d'Ondes Numérique fenêtre, vous pouvez afficher l'activité du analogique sur un seul pin à la fois.



Cliquez pour afficher la page vers la gauche ou la droite, ou utilisez les touches Home, PgUp, PgDn, End.

Vous pouvez **saut** la les lignes du curseur bleu ou rouge jusqu'au prochain "point d'inclinaison" croissant ou décroissant à l'aide des boutons fléchés avant ou arrière ( ,  ou  ,  , à nouveau en fonction du curseur activé, ou utilisez les touches ← et → touches) de concert avec les touches de sélection de la pente montante / descendante  ,  (le "point d'inclinaison" se produit lorsque la tension analogique passe par le seuil haut du niveau logique KmeEnregistrer de l'ATmega pin). Vous pouvez également cliquer à nouveau pour sauter ou faire glisser ces lignes de curseur de la même manière que leur comportement dans le Pin Formes d'Ondes Numérique. fenêtre

Pressage '**Ctrl-S**' à l'intérieur **soit le fenêtre vous permet de sauvegarder le forme d'Onde (X, Y) données** à un texte fichier de votre choix, où **X** est en microsecondes du côté gauche, et **Y** est en boulons.

'I/O' Dispositifs

Un certain nombre de dispositifs différents entourent la 'Uno' ou 'Mega' sur le périmètre de la **Volet du Banc de Laboratoire**. "Petit" 'I/O' dispositifs (on vous autorise jusqu'à 16 en tout) réside sur les côtés gauche et droit du Volet. "Large" 'I/O' dispositifs (auquel vous êtes autorisé jusqu'à 8 au total) ont des éléments "actifs" et résident en haut et en bas de la **Volet du Banc de Laboratoire**. Le nombre souhaité de chaque type de 'I/O' dispositif disponible peut être réglé à l'aide du menu **Configurer | 'I/O' Dispositifs**.

Plus petit 'I/O' Dispositifs	Big 'I/O' Dispositifs
Bouton Poussoir: 2	ServoMoteur: 1
Résistance de l'interrupteur: 4	DC Moteur: 1
Haut-parleur Piézo: 2	Moteur Pas à Pas: 1
couleur DEL: 2	Moteur Pas à Pas Pulsé: 1
4-DEL Row: 1	Pulseur Binaire: 1
7 segments DEL: 2	Générateur de Fonction: 1
Fil de Connexion du Pins: 1	SFT Serial: 1
Curseur Analogique: 2	SR Esclave: 1
	1-Wire Esclave: 1
	Un-Tir Générateur: 1
	SPI Esclave: 1
	I2C Esclave: 1
	LCD_SPI de caractère: 1
	LCD_I2C de caractère: 1
	Caractère LCD_D4: 1
	ProgIO UNO: 1

Total (max 16): 16

Total (max 8): 8

Chaque 'I/O' dispositif comporte une ou plusieurs pièces jointes pin représentées par un **deux-chiffre** Numéro pin (00, 01, 02,... 10,11,12, 13 et soit A0-A5, soit 14-19, ensuite) dans une zone d'édition correspondante. Pour les numéros pin, numéros 2 à 9, vous pouvez simplement entrer le numéro chiffre - le 0 initial sera automatiquement indiqué, mais pour pins 0 et 1, vous devez d'abord saisir le 0 initial. Les entrées sont *normalement* sur le côté gauche d'un 'I/O' dispositif, et les sorties sont *normalement* sur la droite (*si l'espace le permet*). Tous les 'I/O' dispositifs répondront directement aux niveaux pin et aux modifications apportées au niveau pin. Ils répondront donc à la bibliothèque fonctions ciblée sur leur pins attaché ou à programmé. 'digitalWrite()' (pour une

opération "bit-banged").

Vous pouvez connecter plusieurs dispositifs au même ATmega pin tant que cela ne crée pas de **conflit électrique**. Un tel conflit peut être créé soit par un 'Uno' ou 'Mega' pin comme 'OUTPUT' Conduite contre un dispositif connecté à forte conduction (basse impédance) (par exemple, conduite contre une sortie 'FUNCGEN' ou un 'MOTOR' Enc sortie), ou par deux dispositifs connectés se faisant concurrence (par exemple un bouton 'PULSER' et un bouton 'PUSH' associé au même pin). Un tel conflit serait désastreux dans une implémentation matérielle réelle et est donc interdit, et sera signalé à l'utilisateur via une boîte de message contextuelle).

La boîte de dialogue peut être utilisée pour permettre à l'utilisateur de choisir les types et les nombres du 'I/O' dispositifs souhaité. De cette boîte de dialogue, vous pouvez également **Enregistrer** 'I/O' dispositifs à un texte fichier et / ou **Chargez** 'I/O' dispositifs à partir d'un texte précédemment enregistré (ou modifié) fichier (**y compris toutes les connexions pin, les paramètres cliquables et toutes les valeurs de la zone de saisie saisie**).

Notez qu'à partir de la version 1.6, le suffixe 'S' (ou 's') peut être attribué aux valeurs des zones d'édition Période, Délai et Largeur d'impulsion de l'I/O dispositifs concerné. Cela indique qu'ils devraient être escaladé en fonction de la position d'un global 'I/O ____S' contrôle de curseur qui apparaît dans le fenêtre principal **Barre d'outils**. Avec ce curseur complètement à droite, le facteur d'échelle est 1.0 (unité), et avec le curseur complètement à gauche, le facteur d'échelle est 0.0 (sous réserve des valeurs minimales imposées par chaque 'I/O' dispositif). Vous pouvez mettre à l'échelle plus d'une valeur de zone d'édition **simultanément** en utilisant ce curseur. Cette fonctionnalité vous permet de faire glisser le curseur tout en exécutant pour émuler facilement des largeurs d'impulsions, des périodes et des délais changeants pour les utilisateurs connectés 'I/O' dispositifs.

Le reste de cette section fournit une description de chaque type de dispositif.

Plusieurs de ces dispositifs **prend en charge la mise à l'échelle de leurs valeurs saisies** en utilisant le curseur sur la fenêtre principal **Barre d'outils**. Si la valeur dispositif a pour suffixe la lettre 'S', sa valeur sera multipliée par un

facteur d'échelle (compris entre 0,0 et 1,0) déterminé par la position du curseur du curseur, sous réserve de la contrainte de valeur minimale dispositif. (1.0 est complètement à droite, 0.0 est complètement à gauche) --voir 'I/O ____S' sous chacun des tuyaux

dispositifs détaillés ci-dessous.



'Serial' Moniteur ('SERIAL')

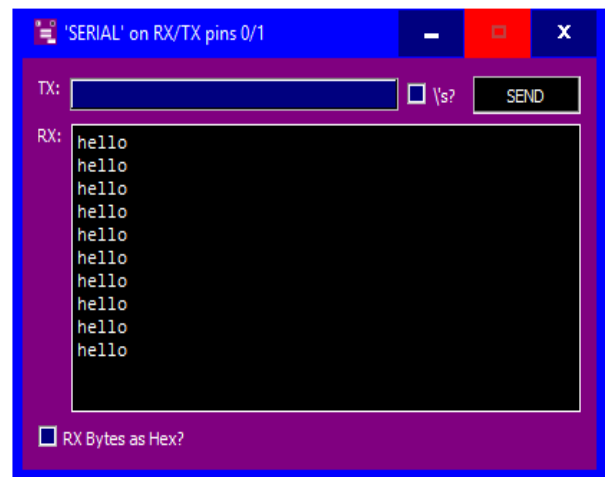


Ce 'I/O' dispositif permet l'entrée et la sortie série via le matériel ATmega (via la puce USB 'Uno' ou 'Mega') sur les 'Uno' ou 'Mega' pins 0 et 1. Le débit en bauds est défini à l'aide de la liste déroulante en bas - le débit en bauds sélectionné **doit correspondre** la valeur que votre programme transmet au '`Serial.begin()`' fonction pour une transmission / réception correcte. La communication série est fixée à 8 bits de données, 1 bit d'arrêt et aucun bit de parité. Tu es autorisé à **déconnecter** (blanc) **mais pas remplacer** TX pin 00, mais pas RX pin 01.

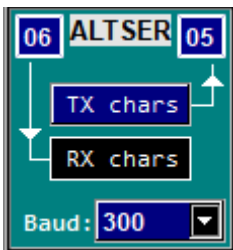
Pour envoyer une entrée au clavier sur votre programme, saisissez un ou plusieurs caractères dans la partie supérieure (caractères de transmission), puis modifiez-le. frapper le '**Enter**' touche du clavier. (les caractères deviennent en italique pour indiquer que les transmissions ont commencé) - ou, s'ils sont déjà en cours, les caractères dactylographiés ajoutés seront en italique. Vous pouvez ensuite utiliser le '`Serial.available()`' et '`Serial.read()`' fonctions pour lire les caractères dans l'ordre dans lequel ils ont été reçus dans la mémoire tampon pin 0 (le caractère saisi le plus à gauche sera envoyé en premier). Des impressions textuelles et numériques formatées, ou des valeurs d'octet non formatées, peuvent être envoyées à la sortie de console inférieure (caractères RX) fenêtre en appelant l'Arduino. '`print()`', '`println()`', ou '`write()`' fonctions.

En outre, **un fenêtre plus grand pour régler / visualiser les caractères TX et RX peut être ouvert en double-cliquant (ou en faisant un clic droit) sur cette 'SERIAL' dispositif**.

Ce nouveau fenêtre a une boîte d'édition de caractères TX plus grande et un bouton 'Send' séparé sur lequel il est possible de cliquer pour envoyer les caractères TX au 'Uno' ou 'Mega' (sur le pin 0). Il existe également une option de case à cocher pour réinterpréter les séquences de caractères d'échappement avec une barre oblique inverse, telles que '`\n`' ou '`\t`' pour un affichage non brut.



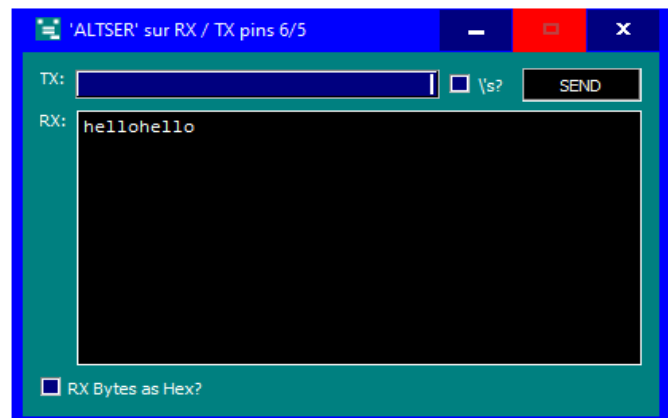
Alterner Série ('ALTSER')



Ce 'I/O' dispositif permet la bibliothèque, ou, alternativement, l'entrée et la sortie série "bit-bang" de l'utilisateur sur n'importe quelle paire de 'Uno' ou 'Mega' pins que vous choisissiez de remplir (à l'exception de pins 0 et 1 dédiés au matériel 'Serial' la communication). Votre programme doit avoir un '`#include <SoftwareSerial.h>`' ligne vers le haut si vous souhaitez utiliser les fonctionnalités de cette bibliothèque. Comme avec le 'SERIAL' dispositif, le débit en bauds pour 'ALTSER' est défini à l'aide de la liste déroulante en bas - le débit en bauds sélectionné doit correspondre à la valeur que votre programme transmet au '`begin()`' fonction pour une transmission / réception correcte. La communication série est fixée à 8 bits de données, 1 bit d'arrêt et aucun bit de parité.

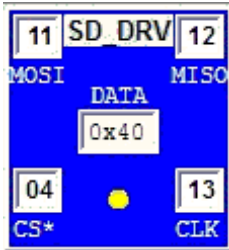
En outre, comme avec le matériel basé 'SERIAL', **Un fenêtre plus grand pour le paramétrage / la visualisation TX et RX peut être ouvert en double-cliquant (ou en faisant un clic droit) sur le ALTSER dispositif**.

Notez que contrairement à la mise en œuvre matérielle de 'Serial', il n'y a pas fourni Mémoire tampon prise en charge par les opérations d'interruption internes ATmega (uniquement une mémoire tampon RX), donc cette '`write()`' (ou '`print()`') les appels sont bloquants (c'est-à-dire que votre programme ne sera pas traité tant qu'ils ne sont pas terminés).



Lecteur de Disque SD ('SD_DRV')

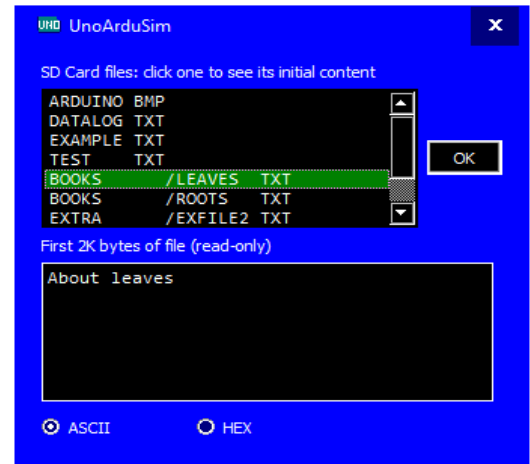
Ce 'I/O' dispositif permet d'utiliser une bibliothèque à médiation logicielle (mais **ne pas** "bit-banged") Entrées et sorties du fichier sur le 'Uno' ou 'Mega' **SPI** pins (vous pouvez choisir lequel **CS *** pin que vous utiliserez). Votre programme peut simplement `#include <SD.h>` ligne près du sommet, et vous pouvez utiliser `<SD.h>` fonctions OU appeler directement `SdFile` fonctions vous-même.



Un plus grand fenêtre affichant les répertoires et le fichiers (et le contenu) peuvent être ouverts en double-cliquant (ou en faisant un clic droit) sur le 'SD_DRV' dispositif. . Tout le contenu du disque est **chargé de** un **Dakota du Sud** sous-répertoire dans le répertoire programme chargé (s'il existe) à `'SdVolume::init()'`, **et se reflète dans** cela même **Dakota du Sud** sous-répertoire sur fichier `'close()'`, `'remove()'`, et sur

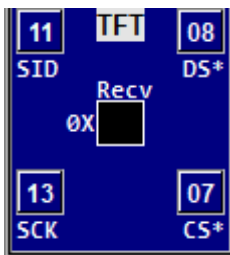
`'mkdir()'` et `'rmdir()'`.

Un DEL jaune clignote pendant les transferts SPI et 'DATA' affiche le dernier message 'SD_DRV'. **réponse** octet. Tous les signaux SPI sont précis et peuvent être visualisés dans un **Forme d'Onde** fenêtre.



Écran TFT ('TFT')

Cette 'I/O' dispositif émule un Adafruit™ affichage TFT de taille 1280by-160 pixels (dans sa rotation native = 0, mais pour l'utilisation de la bibliothèque de 'TFT.h', `'TFT.begin()'` ensembles d'initialisation pour rotation = 1, qui donne une vue "paysage" de 160-par-128 pixels). Vous pouvez conduction dispositif ce en appelant le fonctions du `'TFT.h'` bibliothèque (qui nécessite d'abord `#include <TFT.h>`), ou vous pouvez utiliser le système SPI pour vous envoyer des séquences d'octets propres à conduction il.

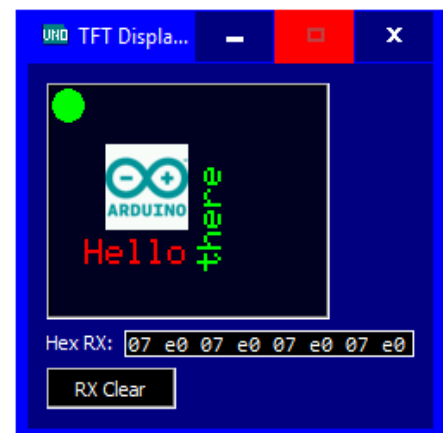


Le TFT est toujours connecté à 'SPI' 'MOSI' pins (pour 'SID') et 'SCK' (pour 'SCK') - ceux qui ne peuvent être modifiés. Le 'DS*' pin est pour données / commande de sélection (mode de données de 'LOW') et la 'CS*' pin est le puce-select actif-bas

Il n'y a pas Réinitialiser pin fourni pour que vous ne pouvez pas faire une réinitialisation matérielle ce dispositif en entraînant un faible pin (comme le `'TFT::begin()'` fonction tente de faire quand

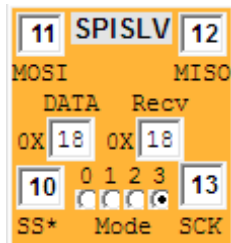
vous avez passé un numéro valide 'reset' pin en tant que troisième paramètre à la `'TFT(int cs, int ds, int rst)'` constructeur). Le dispositif a cependant une connexion cachée au système ligne Réinitialiser il se remet à zéro à chaque fois que vous cliquez sur le principal UnoArduSim Réinitialiser icône de la barre d'outils, ou le bouton de réinitialisation 'Uno' ou 'Mega' circuit imprimé ..

Par **double-clic** (ou **clic droit**) Sur ce dispositif, une plus grande fenêtres est ouverte pour montrer que le plein 160 par 128 pixels écran à cristaux liquides, ainsi que les plus récemment reçu 8 octets (comme indiqué ci-dessous)



SPI Esclave Configurable ('SPISLV')

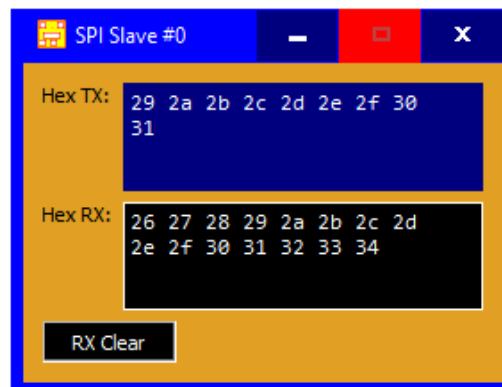
Ce dispositif de 'I/O' émule un esclave SPI en mode choisi avec un actif bas **SS *** ("slave-select") pin contrôlant le **MISO** sortie pin (lorsque **SS *** est haut, **MISO** n'est pas conduite). Votre programme doit avoir un `'#include <SPI.h>'` Si vous souhaitez utiliser les fonctionnalités du intégré SPI Arduino objet et de la bibliothèque. Alternativement, vous pouvez choisir de créer votre propre "bit-banged" **MOSI** et **SCK** des signaux à conduction ce dispositif.



Le dispositif détecte les transitions de bord sur son **CLK** entrée en fonction du mode sélectionné (`'MODE0'`, `'MODE1'`, `'MODE2'`, ou `'MODE3'`), qui doivent être choisis pour correspondre au mode programmé SPI de votre programme.

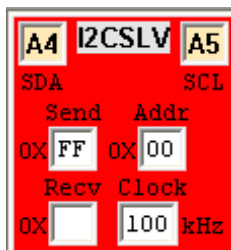
En double-cliquant (ou en faisant un clic droit) sur le dispositif, vous pouvez ouvrir un compagnon plus grand, le fenêtre. que la

place permet y Vous devez remplir un tampon de 32 octets maximum (pour émuler SPI dispositifs qui renverra automatiquement leurs données) et voir les 32 derniers octets reçus (tous sous forme de paires hexadécimales). **Notez que** l'octet tampon TX suivant est envoyé automatiquement à 'DATA' uniquement après plein `'SPI.transfer()'` a complété!



I2C Esclave TwoWire ('I2CSLV')

Ce 'I/O' dispositif émule seulement une *mode esclave* dispositif. Le dispositif peut se voir attribuer une adresse de bus I2C à l'aide d'une entrée à deux hexs-chiffre dans sa boîte de dialogue 'Addr' (il ne répondra qu'à la I2C transactions de bus impliquant son adresse attribuée). Le dispositif envoie et reçoit des données sur son drain ouvert (pull-down-only) **SDA** pin, et répond au signal d'horloge de bus sur son drain ouvert (pull-down-only) **SCL** pin. Bien que le 'Uno' ou 'Mega' soit le maître de bus responsable de la génération du **SCL** signal, cet esclave dispositif tirera également **SCL** bas pendant sa phase basse afin d'étendre (s'il le faut) le temps bas du bus à un temps adapté à sa vitesse interne (pouvant être réglé dans sa boîte de dialogue 'Clock').



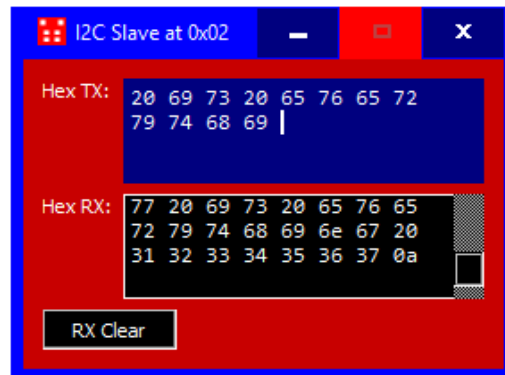
Votre programme doit avoir un `'#include <Wire.h>'` si vous souhaitez utiliser les fonctionnalités du `'TwoWire'` bibliothèque pour interagir avec ce dispositif. Alternativement, vous pouvez choisir de créer vos propres données bit-bangées et signaux d'horloge pour conduction cet esclave dispositif.

Un seul octet à renvoyer au maître 'Uno' ou 'Mega' peut être défini dans la boîte de dialogue 'Send' et un seul octet (reçu le plus récemment) peut être visualisé dans sa boîte de dialogue 'Recv' (en lecture seule). **Notez que** la valeur de la zone d'édition 'Send' reflète toujours la suivant octet pour la transmission à partir de ce tampon de données interne dispositif.

En double-cliquant (ou en faisant un clic droit) sur le dispositif, vous pouvez ouvrir un compagnon plus grand, le fenêtre. qui

vous permet plutôt de remplir une mémoire tampon FIFO maximale de 32 octets (afin d'émuler le TWI dispositifs avec une telle fonctionnalité) et d'afficher (jusqu'à un maximum de 32) octets des dernières données reçues (sous forme de deux fichiers). hex-chiffre affichage de 8 octets par ligne). Le nombre de lignes dans ces deux zones d'édition correspond à la taille de la mémoire tampon TWI choisie (pouvant être sélectionnée à l'aide de **Configurer | Préférences**). Ceci a été ajouté en option depuis l'Arduino `'Wire.h'` utilisations de la bibliothèque **cinq** tels tampons de RAM dans son code de mise en œuvre, ce qui coûte cher en mémoire

RAM. En éditant l'installation Arduino `'Wire.h'` fichier pour changer la constante définie `'BUFFER_LENGTH'` (et aussi l'édition du compagnon `'utility/twi.h'` fichier pour changer Longueur de tampon TWI) à la place soit 16 ou 8, un utilisateur *pourrait* réduire de manière significative la surcharge de mémoire RAM de le 'Uno' ou 'Mega' dans un ciblé **implémentation matérielle** - UnoArduSim reflète donc cette possibilité réelle grâce à **Configurer | Préférences**

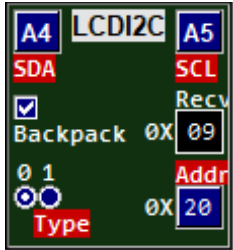


Texte LCD I2C ('LCDI2C')

Cette 'I/O' dispositif émule un 1,2, o4 4-line caractère LCD, dans l'un des trois modes:

- a) à dos tapez 0 (extension du port de style Adafruit avec du matériel ayant l'adresse de bus I2C 0x20-0x27)
- b) le type de sac à dos 1 (port style DFRobot extension avec du matériel ayant bus I2C adresse 0x20-0x27)
- c) aucun sac à dos (mode natif interface I2C intégré ayant l'adresse de bus I2C 0x3C-0x3F)

Support code bibliothèque pour chaque mode dispositif a été fournie à l'intérieur du 'include_3rdParty' dossier de votre répertoire d'installation UnoArduSIm: 'Adafruit_LiquidCrystal.h' , 'DFRobot_LiquidCrystal.h' , et 'Native_LiquidCrystal.h' , respectivement.



Le dispositif peut se voir attribuer une adresse de bus I2C en utilisant une entrée à deux hex-chiffre dans son édition boîte de 'Addr' (il ne répondra à I2C transactions de bus impliquant son adresse attribuée). Le dispositif reçoit l'adresse du bus et les données (et réagit avec ACK = 0 ou NAK = 1) sur son drain ouvert (pull-down uniquement) SDA pin. Vous ne pouvez écrire des commandes LCD et données DDRAM - vous **ne peut pas** relire les données des emplacements de DDRAM

écrits ..



Double-cliquez sur ou **clic-droit** pour ouvrir l'écran LCD moniteur fenêtres à partir duquel vous pouvez également définir la taille de l'écran et le jeu de caractères.

Texte LCD SPI ('LCDSPI')

Cette 'I/O' dispositif émule un 1,2, o4 4-line caractère LCD, dans l'un des deux modes:

- a) sac à dos (Extension du port de style SPI Adafruit)
- b) sans sac à dos (mode natif intégré SPI Interface - comme indiqué ci-dessous)

Support code bibliothèque pour chaque mode dispositif a été fournie à l'intérieur du 'include_3rdParty' dossier de votre répertoire d'installation UnoArduSIm: 'Adafruit_LiquidCrystal.h' ,, et 'Native_LiquidCrystal.h' , respectivement.



Pin 'SID' sont des données en série dans, 'SS' est le dispositif-sélectionner actif-bas, 'SCK' est l'horloge pin et 'RS' sont les données / commande pin. Vous ne pouvez écrire des commandes LCD et les données DDRAM (toutes les transactions SPI sont des écritures) - vous **ne peut pas** relire les données des emplacements de DDRAM écrits.

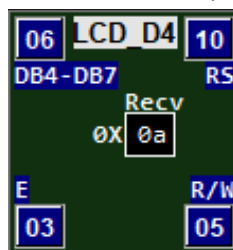
Double-cliquez sur ou **clic-droit** pour ouvrir l'écran LCD moniteur fenêtres à partir duquel vous pouvez également définir la taille de l'écran et le jeu de caractères.



Texte LCD D4 ('LCD_D4')

Cette 'I/O' dispositif émule un 1,2, o4 4-line caractère LCD ayant une interface de bus parallèle 4 bits. octets de données sont écrites / lecture dans **deux moitiés** sur ses 4 pins données 'DB4-DB7' (où la zone de saisie contient la **le plus petit numéro des 4 nombres consécutifs pin**), - les données sont cadencées sur les fronts descendants du 'E' (activé) pin, avec direction de données commandé par le 'R/W' pin, et les données LCD / mode de commande par le 'RS' pin.

Support code bibliothèque a été fournie à l'intérieur du 'include_3rdParty' dossier de votre répertoire d'installation UnoArduSIm:



'Adafruit_LiquidCrystal.h' , , et 'Native_LiquidCrystal.h' travaillent tous les deux.

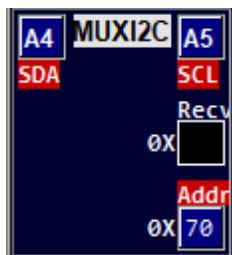
Double-cliquez sur ou **clic-droit** pour ouvrir l'écran LCD moniteur fenêtres à partir duquel vous pouvez également définir la taille de l'écran et le jeu de



caractères.

Multiplexeur I2C DEL ('MUXI2C')

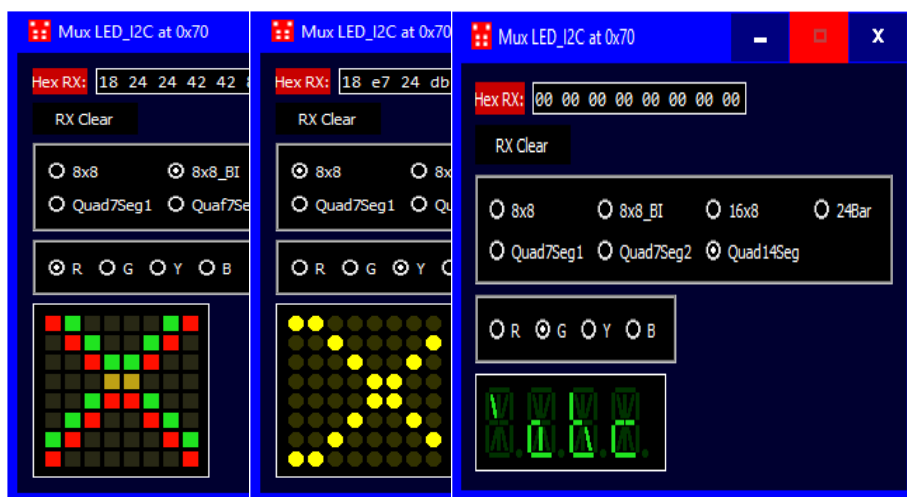
Cette FSM4 contract émule un contrôleur HT16K33 interfacé I2C (h bus I2C yant adresse 0x70-0x77) auquel l'un des différents types de multiplexes DEL écrans peuvent être fixés:



- a) 8x8 ou 16x8 DEL tableau
- b) 8x8 bi-couleur DEL tableau
- c) 24-bi-couleur-DEL bar
- d) deux styles de 4-dispositifs afficheurs 7 segments
- e) une 4-dispositifs 14 segments affichage alphanumérique

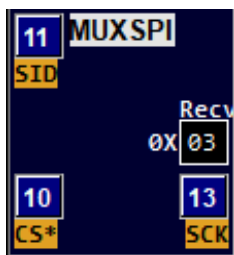
Tous sont pris en charge par le 'Adafruit_LEDBackpack.h' le code prévu à l'intérieur du 'include_3rdParty' dossier:

Double-cliquez sur (Ou clic droit) d'ouvrir un plus grand formes d'Ondes de choisir et vue l'un des plusieurs couleurs DEL affiche.

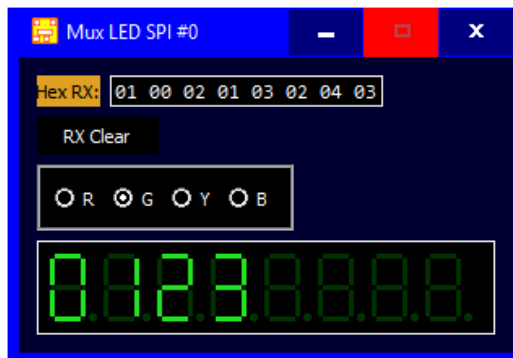


Multiplexeur SPI DEL ('MUXSPI')

A-DEL multiplexés contrôleur sur la base du MAX6219, à l'appui 'MAX7219.h' le code prévu à l'intérieur du dossier 'include_3rdParty' à téléchargement jusqu'à huit chiffres à 7 segments.

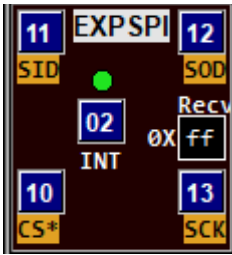


Double-cliquez sur (Ou clic droit) d'ouvrir un plus grand formes d'Ondes regarder le 8-dispositifs coloré Afficheur à 7 segment-.



Port d'Expansion SPI ('EXPSPi')

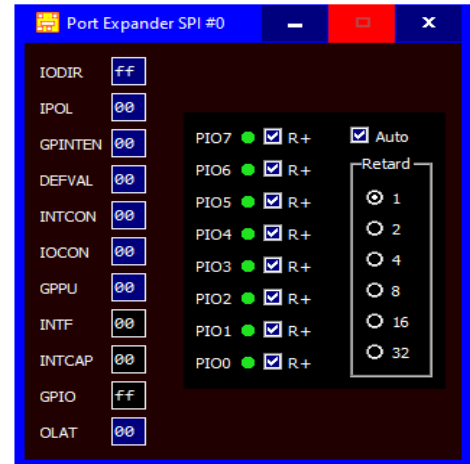
Une 8-bit le port d'extension sur la base du MCP23008, à l'appui 'MCP23008.h' code prévue à l'intérieur du 'include_3rdParty' dossier. Vous pouvez écrire à MCP23008 registres, et lire les GPIO analyseur les niveaux. Interruptions peuvent être activés à chaque changement GPIO analyseur - une interruption déclenchée sera téléchargement le 'INT' analyseur.



Double-cliquez sur (Ou clic droit) ouvrir **une plus grande formes d'Ondes à voir le 8 lignes**

de port GPIO, et les résistances pull-up attachés. Vous pouvez modifier tractions manuellement en cliquant, ou joindre un compteur qui les changent périodiquement de manière de comptage. Le taux auquel les incréments de comptage est déterminé

par le retard à l'échelle vers le bas facteur choisi par l'utilisateur (un emploi correspond facteur 1x à un incrément toutes les 30 millisecondes, les facteurs de retard élevées donnent un taux de comptage plus lent)

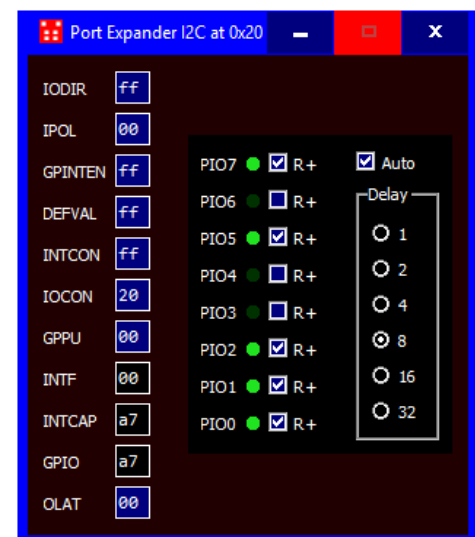


Port d'Expansion I2C ('EXPI2C')

Une 8-bit le port d'extension sur la base du MCP23008, à l'appui 'MCP23008.h' le code prévu à l'intérieur du 'include_3rdParty' dossier. Capacités correspondent aux 'EXPSPi' contract.

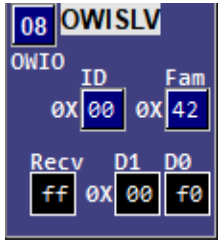


Double-cliquez sur (Ou clic droit) d'ouvrir un plus grand formes d'Ondes comme vient le 'EXPSPi' contract.



'1-Wire' Esclave ('OWIISLV')

Ce dispositif de 'I/O' émule l'un des petits ensembles de bus dispositifs de '1-Wire' connectés à pin OWIO. Vous pouvez créer un bus '1-Wire' (avec un ou plusieurs de ces esclaves '1-Wire' dispositifs) sur le 'Uno' ou 'Mega' pin de votre choix. Ce dispositif peut être utilisé en appelant le '**OneWire.h**' bibliothèque fonctions après avoir placé un '**#include <OneWire.h>**' la ligne en haut de votre programme. Sinon, vous pouvez également utiliser des signaux sur bit sur OWIO avec ce dispositif (bien que ce soit très difficile à faire correctement sans provoquer de conflit électrique - un tel conflit est toujours possible, même avec le '**OneWire.h**' fonctions, mais ces conflits sont rapportés dans UnoArduSim).

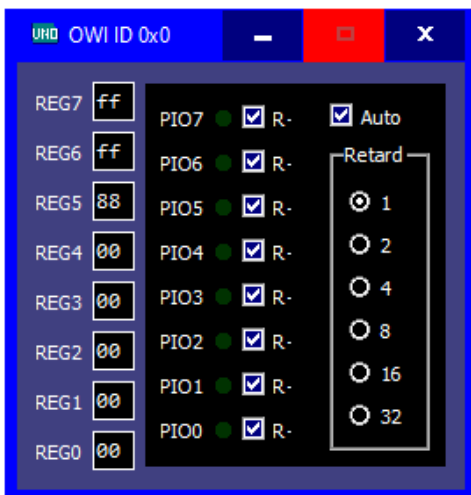


Chaque OWISLV dispositif du monde réel doit avoir un identifiant unique (64 bits!) De 8 octets. numéro de série interne - dans UnoArduSim, il s'agit d'un fichier simplifié grâce à l'utilisateur qui fournit un court hexadécimal sur 1 octet. '**ID**' valeur (affectée séquentiellement par défaut à la charge / ajout de dispositif), plus le '**Fam**' Code de la famille pour ce dispositif. UnoArduSim reconnaît un petit ensemble de codes de famille à partir de V2.3 (0x28, 0x29, 0x3A, 0x42) dispositifs couvrant les capteurs de température et les IO parallèles (PIO) (un code de famille non reconnu définit le dispositif comme un bloc-notes générique de 8 octets) dispositif avec capteur générique.

Si la famille dispositif n'a pas de registres PIO, registres **D0** et **D1** représenter les deux premiers octets du bloc-notes, sinon ils représentent le PIO Le registre "status" (niveaux réels de pin) et le registre de données de verrouillage PIO pin, respectivement.

Par **double-clic** (ou **clic droit**) sur le dispositif, un plus grand **OWIMonitor** fenêtre est ouvert. À partir de ce fenêtre plus grand, vous pouvez inspecter tous les registres dispositif, modifier les emplacements du bloc-notes SCR0 et SCR1 à l'aide de modifications et d'un curseur (encore une fois, SCR0 et SCR1 ne correspondent qu'à **D0** et **D1** si aucun PIO n'est présent), ou définissez des tractions PIO pin externes. Lorsque SCR0 et SCR1 sont modifiés, UnoArduSim mémorise ces valeurs modifiées en tant que "préférence" utilisateur représentant une valeur initiale (à partir de Réinitialiser) représentant la valeur signée générée par le capteur dispositif; s - le curseur est réinitialisé à 100% (un facteur d'échelle de 1,0) au moment de l'édition. Lorsque le curseur est ensuite déplacé, le '**signed**' la valeur dans SCR1 est réduite en fonction de la position du curseur (facteur d'échelle allant de 1,0 à 0,0) - sa fonction vous permet de tester facilement la réponse de votre programme pour une modification en douceur des valeurs de capteur. .

Pour un dispositif avec PIO pins, lorsque vous définissez les cases à cocher de niveau pin, UnoArduSim se souvient de ces valeurs cochées. que les tractions actuelles appliquées à l'extérieur du pins. Ces valeurs externes de soulèvement sont ensuite utilisées avec les données de verrouillage pin (registre **D1**) pour déterminer les niveaux finaux réels du pin et allumer ou éteindre le DEL vert attaché au PIO pin (le pin ne va que '**HIGH**' si un pull-up externe est appliqué, et le correspondant **D1** le loquet est un '1').



Programmable 'I/O' Dispositif ('PROGIO')



Ce 'I/O' dispositif est en fait un 'Uno' circuit imprimé nu que vous pouvez programmer (avec un programme séparé) afin d'émuler un 'I/O' dispositif dont vous pouvez définir complètement le comportement. Vous pouvez choisir jusqu'à quatre pins (IO1, IO2, IO3 et IO4) que cet esclave 'Uno' partagera avec le maître (main 'Uno' ou 'Mega') qui apparaît au milieu de votre **Volet du Banc de Laboratoire**. Comme avec les autres dispositifs, tout conflit électrique entre cet esclave 'Uno' et le maître circuit imprimé sera détecté et signalé. Notez que toutes les connexions sont **directement** câblées, **à l'exception de pin 13** (où une résistance série R-1K est supposée entre les deux pins afin d'éviter un conflit électrique à Réinitialiser). Depuis V2.8, les connexions entre maître et esclave pins sont **mappé** : si le maître est aussi un 'Uno', ce **défaut** la cartographie est une identité (sauf que pin 1 est mappé sur pin 0, et vice-versa pour permettre les communications 'Serial'); sinon si le maître est un 'Mega', les pins 1 et 0 sont à nouveau inversés, **et tout** SPI et TWI pins sont mappés pour une connexion directe entre les sous-systèmes maître et esclave correspondants. Cette **défaut** la cartographie peut être **remplacé** en précisant dans votre **IODevs.txt** fichier une liste explicite de 4 numéros pin principaux qui suit le nom PROGIO programme fichier - si ces valeurs sont -1, c'est la même chose que le mappage par défaut. Pour ce dispositif, les numéros pin sont entrés **sont ceux de l'esclave 'Uno'**. L'image de gauche montre les 4 esclaves pins spécifiés pour son système SPI pins (SS *, MISO, MOSI, SCK) - indépendamment du fait qu'il soit maître 'Uno' ou 'Mega', cela vous permettrait de programmer cet esclave en tant que esclave SPI générique (ou maître) dont vous pouvez définir le comportement par programmation.

Par **double-clic** (ou **clic droit**) sur ce dispositif, une fenêtre plus grande est ouverte pour montrer que cet esclave 'Uno' a son propre **Volet de Code** et associés **Volet de Variables**, tout comme le maître 'Uno' a. Il a aussi son propre **Barre d'outils**. Que vous pouvez utiliser pour **charge** et **contrôle exécution** d'un programme esclave - les actions d'icône ont les mêmes raccourcis clavier que ceux de la fenêtre principale. (**Chargez** est **Ctrl-L**, **Enregistrer** est **Ctrl-S** etc.). Une fois chargé, vous pouvez exécuter à partir de **non plus** Main UnoArduSim fenêtre, ou de l'intérieur de ce moniteur Esclave fenêtre - dans les deux cas, les Main 'Uno' programme et Esclave 'Uno' programme restent bloquées en synchronisation avec le passage du temps réel à mesure que leurs exécutions avancent. **Pour choisir le Volet de Code qui aura le charge, recherche et focus exécution**, Cliquez sur sur la barre de titre de son parent fenêtre - la Volet de Code non focalisée a alors sa barre d'outils actions grisées.

Certaines idées possibles pour l'esclave dispositifs qui pourraient être programmé dans ce 'PROGIO' dispositif sont répertoriées ci-dessous. Pour une émulation série, I2C ou SPI dispositif, vous pouvez utiliser le codage programme approprié avec tableaux pour les mémoires tampons d'envoi et de réception, dans l'ordre. émuler un comportement complexe dispositif:

a) Un maître SPI ou un esclave dispositif. UnoArduSimV2.4.a étendu la '**SPI.h**' bibliothèque pour autoriser le mode esclave S {opération PI à travers un '**mode**' paramètre dans '**SPI.begin(int mode = SPI_MASTR)**'. Passer explicitement '**SPI_SLAVE**' pour sélectionner le mode esclave (au lieu de s'appuyer sur le mode maître par défaut). Vous pouvez également maintenant définir une interruption utilisateur fonction (appelons ça '**onSPI**') dans 'Uno' pour transférer des octets en appelant une autre extension ajoutée '**SPI.attachInterrupt(user_onSPI)**'. **Maintenant** calling '**rxbyte=SPI.transfer(tx_byte)**' de l'intérieur de votre '**user_onSPI**' fonction effacera l'indicateur d'interruption et **revenir immédiatement** avec l'octet que vous venez de recevoir dans votre variable '**rxbyte**'. Sinon, vous pouvez éviter de joindre une interruption SPI, et au lieu de simplement appeler '**rxbyte=SPI.transfer(tx_byte)**' à partir de votre programme principal - cet appel sera **bloc exécution** jusqu'à ce qu'un octet SPI ait été transféré, et sera alors **revenir** avec le nouvel octet reçu à l'intérieur '**rxbyte**'.

b) Un générique **série 'I/O'** dispositif. Vous pouvez communiquer avec le Master circuit imprimé en utilisant soit '**Serial**', ou une '**SoftwareSerial**' défini à l'intérieur de votre esclave programme - pour '**SoftwareSerial**' vous devez définir '**txpin**' et '**rxpin**' opposé à celles du Matser pour que l'esclave reçoive sur le pin sur lequel le maître émet (et vice-versa), mais pour '**Serial**' **uniquement**, les 1 et 0 pins sont déjà retournés pour vous.

c) Un dispositif maître ou esclave 'I2C' générique. L'opération Esclave a maintenant été ajoutée pour compléter la mise en oeuvre de la '**Wire.h**' bibliothèque (fonctions '**begin(address)**', '**onReceive()**' et '**onRequest**' ont maintenant été implémentées afin de prendre en charge les opérations en mode esclave).

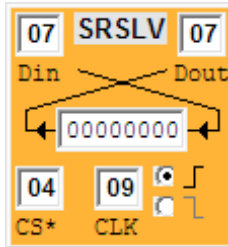
d) Un numérique générique **Pulser**. En utilisant '**delayMicroseconds()**' et '**digitalWrite()**' appels à l'intérieur '**loop()**' dans votre 'PROGIO' programme, vous pouvez définir le '**HIGH**' et '**LOW**' intervalles d'un train

d'impulsions. En ajoutant un séparé `'delay()'` appeler à l'intérieur de votre `'setup()'` fonction, vous pouvez retarder le début de ce train d'impulsions. Vous pouvez même faire varier les largeurs d'impulsion au fil du temps en utilisant un compteur variable. Vous pouvez également utiliser un séparé **'IOx'** pin comme déclencheur pour démarrer le timing d'un '1Shot' émulé (ou à deux coups, à trois coups, etc.) dispositif, et vous pouvez contrôler la largeur d'impulsion produite pour qu'elle change de la manière que vous souhaitez au fil du temps.

e) Un signaleur aléatoire. C'est une variation sur un numérique **Pulser** qui utilise également des appels à `'random()'` et `'delayMicroseconds()'` pour générer des moments aléatoires auxquels `'digitalWrite()'` un signal sur un pin choisi partagé avec le maître. En utilisant les quatre **'IOx'** Le pins autoriserait quatre signaux simultanés (et uniques).

Registre de Décalage Esclave ('SRSLV')

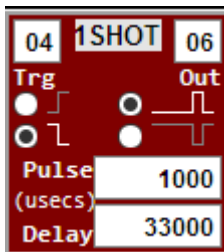
Ce dispositif de 'I/O' émule un simple registre à décalage dispositif avec une valeur active-basse. **SS *** ("slave-select") pin contrôlant le **'Dout'** sortie pin (lorsque **SS *** est haut, **'Dout'** n'est pas conduite). Votre programme pourrait utiliser les fonctionnalités du intégré SPI Arduino objet et de la bibliothèque. Alternativement, vous pouvez choisir de créer votre propre "bit-banged" **'Din'** et **CLK** des signaux à conduction ce dispositif.



Le dispositif détecte les transitions de bord sur son **CLK** entrée qui déclenche le décalage de son registre - la polarité du capteur détecté **CLK** Le bord peut être choisi à l'aide d'un bouton radio. Sur tout **CLK** bord (de la polarité détectée), le registre enregistre sa **Vacarme** niveau dans la position du bit le moins significatif (LSB) du registre à décalage, car les bits restants sont décalés simultanément d'une position à gauche vers la position du bit de poids fort. N'importe quand **SS *** est faible, la valeur actuelle dans la position MSB du registre à décalage est conduite sur **'Dout'**.

Générateur Un-Tir ('1SHOT')

Ce dispositif de 'I/O' émule un one-shot numérique capable de générer une impulsion de polarité et de largeur d'impulsion choisies sur son 'Out' pin, survenant après un délai spécifié par un front de déclenchement reçu sur son **Trg** (déclencheur) entrée pin. Une fois que le front de déclenchement spécifié est reçu, timing commence et une nouvelle impulsion de déclenchement ne sera pas reconnue avant que le 'Out' le poulx a été produit (et est complètement terminé).

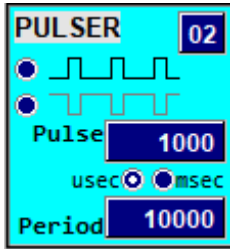


Une utilisation possible de ce dispositif est de simuler des capteurs de distance à ultrasons qui génèrent une impulsion de distance en réponse à une impulsion de déclenchement. Il peut également être utilisé partout où vous souhaitez générer un signal d'entrée pin synchronisé (après le délai que vous avez choisi) sur un signal de sortie pin créé par votre programme. **'Pulse'** et les valeurs 'Delay' peuvent être mises à l'échelle à partir du fenêtre principal **Barre d'outils** Contrôle du curseur de facteur d'échelle 'I/O ____ S' en ajoutant le suffixe 'S' (ou 's') à l'un (ou les deux).

Une autre utilisation de ce dispositif est de tester un programme qui utilise des interruptions, et vous voudriez voir ce qui se passe si un **instruction spécifique programme** est interrompu. Déconnectez temporairement le 'I/O' Dispositif que vous avez connecté à pin 2 (ou pin 3) et remplacez-le par un '1SHOT' dispositif dont le 'Out' pin est connecté à 'pin 2 (ou pin3, respectivement). Vous pouvez ensuite déclencher l'entrée L244. (en supposant que la sensibilité du front montant soit définie ici) en insérant la paire d'instruction `{ 'digitalWrite(LOW) ', 'digitalWrite(HIGH) ' }` **juste avant** à l'instruction dans laquelle vous souhaitez que l'interruption se produise. Réglez le 'Delay' du 1SHOT; pour synchroniser l'impulsion produite sur le 'Out' dans l'instruction programme qui suit cette paire d'instructions de déclenchement. Notez que certaines instructions masquent les interruptions (telles que `'SoftwareSerial.write(byte)'`, unnd so ne peut pas être interrompu.

Pulseur Binaire ('PULSER')

Cette 'I/O' dispositif émule d'un simple générateur d'impulsions formes d'Ondes numérique qui produit un signal périodique qui peut être appliqué à tout choisi 'Uno' ou 'Mega' pin.



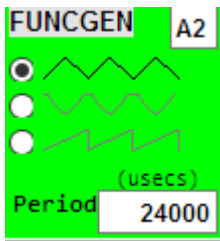
La période et à des impulsions (en microsecondes) peuvent être réglés à l'aide edit-boîtiers de la période minimale autorisée est de 50 microsecondes, et l'impulsion de largeur minimum est de 10 microsecondes. Vous pouvez choisir entre les valeurs type-casté en microsecondes ('usec') et millisecondes ('msec') et sera sauvé ce choix ainsi que les autres valeurs lorsque vous 'Save' du Configurer | 'I/O' Dispositifs.

La polarité peut également être choisi: soit des impulsions positives de pointe (0 à 5 V) ou négative des impulsions de pointe (5V à 0V).

Les valeurs 'Pulse' et 'Period' peuvent être mis à l'échelle de la principale **Outil-Bar** le curseur de facteur d'échelle 'I/O____S' en ajoutant un suffixe 'S' (ou 's') à l'une (ou les deux). Cela vous permet de changer alors la valeur 'Pulse' ou 'Period' **dynamiquement** pendant exécution.

Analogique Générateur de Fonction ('FUNCGEN')

Ce 'I/O' dispositif émule un simple générateur analogique forme d'Onde qui produit un signal périodique pouvant être appliqué à n'importe quel 'Uno' ou 'Mega' pin choisi.



La période (en microsecondes) peut être définie à l'aide de la zone d'édition. La période minimale autorisée est de 100 microsecondes. Le forme d'Onde qu'il crée peut être choisi comme étant sinusoïdal, triangulaire ou en dents de scie (pour créer une onde carrée, utilisez plutôt un 'PULSER'). À des périodes plus courtes, moins d'échantillons par cycle sont utilisés pour modéliser la forme d'Onde produit (seulement 4 échantillons par cycle à une période = 100 microsecondes).

Le 'Period' la valeur peut être mise à l'échelle à partir du fenêtre principal **Barre d'outils** Curseur de facteur d'échelle 'I/O____S' en ajoutant comme suffixe la lettre 'S' (ou 's').

Moteur Pas à Pas ('STEPR')

Ce dispositif de 'I/O' émule un moteur pas à pas bipolaire ou unipolaire de 6 V avec un contrôleur conducteur intégré **soit deux** (sur **P1** , **P2**) **ou quatre** (sur **P1** , **P2** , **P3** , **P4**) signaux de commande. Le nombre de pas par tour peut également être défini. Vous pouvez utiliser le '**Stepper.h**' fonctions '**setSpeed()**' et '**step()**' à conduction le 'STEPR'. Alternativement, 'STEPR' *aussi répondre* à la vôtre '**digitalWrite()**' "bit-banged" conduction.



Le moteur est modélisé avec précision à la fois mécaniquement et électriquement. Les chutes de tension du moteur conducteur et les variations de réluctance et d'inductance sont modélisées avec un moment d'inertie réaliste par rapport au couple de maintien.

L'enroulement du rotor du moteur a une résistance modélisée de $R = 6$ ohms et une inductance de $L = 6$ milli-Henries, ce qui crée une constante de temps électrique de 1,0 milliseconde. En raison de la modélisation réaliste, vous remarquerez que les impulsions de commande pin très étroites *ne soyez pas* le moteur à marcher - à la fois en raison du temps de montée du courant fini et de l'inertie du rotor. Cela concorde avec ce qui est observé lors de la

conduite d'un moteur pas à pas réel à partir d'un 'Uno' ou 'Mega' avec, bien sûr, un modèle approprié (**et requis**) moteur conducteur puce entre les fils du moteur et le 'Uno' ou 'Mega'!

Un malheureux bogue dans l'Arduino '**Stepper.h**' Le code de bibliothèque signifie que, lors de la réinitialisation, le moteur pas à pas ne sera pas en position Un Pas 1 (sur quatre étapes). Pour surmonter cela, l'utilisateur doit utiliser '**digitalWrite()**' dans son '**setup()**' routine pour initialiser les niveaux de contrôle pin au '**step(1)**' niveaux appropriés pour le contrôle 2-pin (0,1) ou 4-pin (1,0,1,0) et permettre au moteur de se déplacer vers la position souhaitée initiale du moteur pendant 10 millisecondes.

AS de V2.6, ce dispositif comprend maintenant un 'sync' DEL (vert pour synchroniser, ou rouge lorsqu'il est éteint par

une ou plusieurs étapes) .Also, en plus du nombre de pas par tour, deux valeurs supplémentaires (cachées) peut éventuellement indiqué dans les IODEvs.txt fichier pour spécifier le load-mécanique par exemple, les valeurs 20, 50, 30 spécifie 20 pas par tour, un couple de charge d'inertie 50 fois celle du rotor de moteur lui-même, et un couple de charge de 30 pour cent de plein couple de maintien du moteur.

Notez que **la réduction de vitesse n'est pas directement prise en charge** manque d'espace, mais vous pouvez l'émuler dans votre programme en implémentant un compteur modulo-N variable et en appelant uniquement 'step()' lorsque ce compteur atteint 0 (pour la réduction de vitesse par le facteur N).

Moteur Pas à Pas Pulsé ('PSTEPR')

Cette 'I/O' dispositif émule un 6V **micro-pas à pas** bipolaire Moteur Pas à Pas avec un contrôleur intégré conducteur conduite par un '**Step**' pulsé pin, un bas-actif '**EN***' (Activé) pin, et un '**DIR**' (direction) pin . Le nombre d'étapes entières par révolution peut également être réglé directement, ainsi que le nombre de micro-pas par pas entier (1,2,4,8, ou 16). En plus de ces paramètres, deux valeurs supplémentaires (cachées) peut éventuellement indiqué dans les IODEvs.txt fichier pour spécifier le load-mécanique par exemple, les valeurs 20, 4, 50, 30 spécifie 20 pas par tour, 4 micro-pas par pas entier, un moment de charge d'inertie 50 fois celle du moteur rotor lui-même, et un couple de charge de 30 pour cent de la pleine couple de maintien du moteur.

Vous devez écrire du code pour le contrôle conduction pins de façon appropriée.

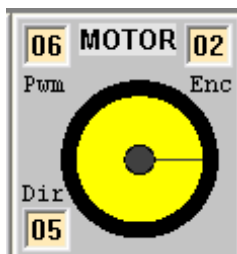


Le moteur est modélisé avec précision à la fois mécaniquement et électriquement. Tension moteur-conducteur gouttes et variation de la réluctance et l'inductance sont modélisés avec un moment d'inertie réaliste par rapport au couple de maintien. L'enroulement du rotor du moteur a une résistance de $R =$ modélisé 6 ohms et une inductance de $L = 6$ milli-Henries qui crée une constante de temps électrique de 1,0 milliseconde.

cette dispositif comprend une activité de 'STEP' jaune DEL, et un 'sync' DEL (vert pour synchroniser, ou rouge lorsqu'il est éteint par une ou plusieurs étapes).

DC Moteur ('MOTOR')

Ce 'I/O' dispositif émule un moteur à courant continu à engrenages 100: 1 alimenté en 6 volts avec un contrôleur conducteur intégré conduite par un signal de modulation de largeur d'impulsion (sur son **Pwm** entrée) et un signal de commande de direction (sur son **Dir** contribution). Le moteur dispose également d'une sortie codeur de roue dont le conduit **Enc** sortie pin. Vous pouvez utiliser '**analogWrite()**' au conduction le **Pwm** pin avec 490 Hz (sur le pins 3,9,10,11) ou 980 Hz (sur le pins 5,6) PWM forme d'Onde avec un rapport cyclique compris entre 0,0 et 1,0 ('**analogWrite()**' valeurs 0 à 255). Alternativement, 'MOTOR' aussi répondre à la vôtre '**digitalWrite()**' "bit-banged" conduction.



Le moteur est modélisé avec précision à la fois mécaniquement et électriquement. La prise en compte des baisses de tension des transistors conducteur et du couple réaliste des engrenages à vide donne une vitesse complète d'environ 2 tours par seconde et un couple de décrochage d'un peu plus de 5 kg-cm (survenant à un cycle de travail PWM constant de 1,0), avec moment d'inertie total moteur-charge de 2,5 kg-cm. Le bobinage du rotor du moteur a une résistance modélisée de $R = 2$ ohms et une inductance de $L = 300$ micro-Henries, ce qui crée une constante de temps électrique de 150 microsecondes. En raison de la modélisation réaliste, vous remarquerez que des impulsions PWM très étroites *ne soyez pas* le moteur à tourner - à la fois en raison du temps de montée du courant fini et du temps d'arrêt significatif

après chaque impulsion étroite. Celles-ci se combinent pour créer un moment de rotation du rotor insuffisant pour vaincre les retours de type ressorts de la boîte de vitesses sous l'effet d'un frottement statique. La conséquence est lors de l'utilisation '**analogWrite()**', un cycle de travail inférieur à environ 0,125 ne fera pas bouger le moteur - cela correspond à ce qui est observé lors de la conduite d'un véritable motoréducteur à partir d'un 'Uno' ou 'Mega' avec, bien sûr, un modèle approprié (**et requis**) module moteur conducteur entre le moteur et le 'Uno' ou 'Mega'!

Le codeur de moteur émulé est un capteur d'interruption optique monté sur l'arbre qui produit un facteur de charge forme d'Onde à 50% ayant 8 périodes complètes de hauteur par tour (pour que votre programme puisse détecter les changements de rotation de la roue avec une résolution de 22,5 degrés).

ServoMoteur ('SERVO')

Ce 'I/O' dispositif émule un servomoteur CC à alimentation 6 volts PWM-conduite à commande de position. Les paramètres de modélisation mécaniques et électriques pour le fonctionnement du servo correspondront étroitement à ceux d'un servo standard HS-422. Le servo a une vitesse de rotation maximale d'environ 60 degrés en 180 millisecondes. Si la case à gauche est cochée, le servo devient un **rotation continue** servo avec la même vitesse maximale, mais maintenant la largeur d'impulsion PWM définit la **la vitesse** plutôt que l'angle



Votre programme doit avoir un `'#include <Servo.h>'` ligne avant de déclarer votre `'Servo'` les instances) si vous choisissez d'utiliser la fonctionnalité de bibliothèque `'Servo.h'`, par exemple `'Servo.write()'`, `'Servo.writeMicroseconds()'` Alternativement, `'SERVO'` répond également à `'digitalWrite()'` Signaux "bit-banged". En raison de la mise en œuvre interne de UnoArduSim, vous êtes limité à 6 `'SERVO'` dispositifs.

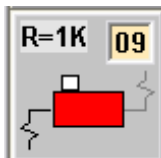
Haut-parleur Piézo ('PIEZO')



Ce dispositif vous permet "d'écouter" les signaux sur le pin 'Uno' ou 'Mega' choisi et peut être un complément utile aux voyants permettant de déboguer votre programme. Vous pouvez également vous amuser un peu en jouant des sonneries. `'tone()'` et `'delay()'` appels (bien qu'il n'y ait pas de filtrage du forme d'Onde rectangulaire, vous n'entendrez donc pas de notes "pures").

Vous pouvez également écouter un 'PULSER' ou un 'FUNCGEN' dispositif connecté en raccordant un 'PIEZO' au pin avec un dispositif conduit.

Résistance de glissière ('R=1K')



Ce dispositif permet à l'utilisateur de se connecter à un pin 'Uno' ou 'Mega', soit une résistance de charge de 1 kohm à + 5V, soit de résistance de 1 kohm à la terre. Cela vous permet de simuler des charges électriques ajoutées à un matériel dispositif réel. En faisant un clic gauche sur le curseur **corps** vous pouvez activer ou désactiver la sélection souhaitée. Utiliser un ou plusieurs de ces dispositifs vous permettrait de définir un "code" à un (ou plusieurs) bits pour que votre programme puisse lire et répondre.

Bouton Poussoir ('PUSH')



Ce 'I/O' dispositif émule un disque normalement ouvert **verrouillage momentané OU** Bouton-poussoir unipolaire, à une portée (SPST) avec une résistance de montée ou baisse de 10 kohms. Si une sélection de transition de front montant est choisie pour le dispositif, les contacts du bouton-poussoir seront câblés entre le dispositif pin et le + 5 V, avec un abaissement à la masse de 10 k Ohms. Si une transition de front descendant est choisie pour le dispositif, les contacts du bouton-poussoir seront câblés entre le dispositif pin et la masse, avec un pull up de 10 k Ohms jusqu'à + 5V.

En cliquant sur le bouton ou en appuyant sur une touche quelconque, vous fermez le contact du bouton-poussoir. Dans **momentané** mode, il reste fermé aussi longtemps que vous maintenez enfoncé le bouton ou la souris, et **loquet** mode (activé en cliquant sur le 'latch' bouton), il reste fermé (et de couleur différente) jusqu'à ce que vous appuyiez à nouveau sur le bouton. Le rebond des contacts (pendant 1 milliseconde) sera produit chaque fois que vous Utilisez le **barre d'espace** appuyer sur le bouton-poussoir.

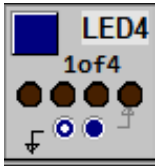
DEL coloré ('LED')



Vous pouvez connecter un DEL entre le pin 'Uno' ou 'Mega' choisi (par l'intermédiaire d'une résistance de limitation de courant série intégré cachée 1 k Ohm) à la terre ou au + 5V. Vous avez ainsi le choix de laisser le DEL s'allumer lorsque le Un Pas Franchir 'Uno' ou 'Mega' connecté est allumé. 'HIGH' ou plutôt lorsqu'il est 'LOW'.

La couleur du DEL peut être choisie en rouge ('R'), en jaune ('Y'), en vert ('G') ou en bleu ('B') à l'aide de sa zone d'édition.

4-DEL Rangée ('LED4')



Vous pouvez connecter cette rangée de 4 DEL colorées entre l'ensemble choisi de 'Uno' ou 'Mega' pins (chacune possède une résistance de limitation de courant série intégré cachée 1 k Ohm) à la terre ou à + 5V - vous avez ainsi le choix d'allumer les DEL. lorsque le 'Uno' ou 'Mega' pin connecté est 'HIGH' ou plutôt lorsqu'il est 'LOW'.

le '1of4' La boîte d'édition pin accepte un seul numéro pin, ce qui signifie **le premier de quatre consécutifs** 'Uno' ou 'Mega' pins qui se connectera aux 4 voyants.

La couleur DEL ('R', 'Y', 'G' ou 'B') est une couleur **option cachée** ça peut être **être choisi par éditer le IODevices.txt fichier** (lequel vous pouvez créer en utilisant **Enregistrer** du **Configurer | 'I/O' Dispositifs** boîte de dialogue).

7 segments DEL Chiffre ('7SEG')



Vous pouvez connecter cet écran Chiffre DEL à 7 segments à un ensemble choisi de **quatre 'Uno' ou 'Mega' pins consécutifs qui donnent le code hexadécimal** pour le chiffre affiché souhaité ('0' à 'F') et activez ou désactivez ce chiffre à l'aide du CS * pin (actif-LOW pour ON).

Ce dispositif comprend un décodeur intégré qui utilise le **actif-HAUT** niveaux sur les quatre consécutives '1of4' pins pour déterminer le hexadécimal chiffre demandé à afficher. Le niveau du plus petit numéro pin (celui affiché dans le '1of4' zone d'édition) représente le bit le moins significatif du code hexadécimal à 4 bits.

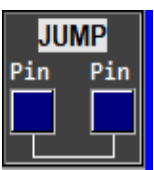
La couleur des segments DEL ('R', 'Y', 'G' ou 'B') est une couleur **option cachée** ça peut être **être choisi par éditer le IODevices.txt fichier** vous pouvez créer en utilisant **Enregistrer** du **Configurer | 'I/O' Dispositifs** boîte de dialogue.

Curseur Analogique

Un potentiomètre 0-5V commandé par curseur peut être connecté à n'importe quel KL165 'Uno' ou 'Mega' choisi pour produire un niveau de tension analogique statique (ou à variation lente) qui sera lu par 'analogRead()' comme valeur de 0 à 1023. Utilisez la souris pour faire glisser, ou cliquez pour sauter, le curseur analogique.



Fil de Connexion du Pins ('JUMP')



Vous pouvez connecter deux 'Uno' ou 'Mega' pins ensemble à l'aide de ce dispositif (si un conflit électrique est détecté lorsque vous remplissez le deuxième numéro pin, la connexion choisie est interdite et le pin est déconnecté).

Ce cavalier dispositif a une utilité limitée. Il est particulièrement utile lorsqu'il est combiné à des interruptions pour les tests programme, l'expérimentation et des fins d'apprentissage. **À partir de UnoArduSim V2.4, l'utilisation d'un 'PROGIO' dispositif offre peut-être plus de flexibilité que les méthodes interruption-conduite ci-dessous..**

Les trois utilisations possibles de ce dispositif sont les suivantes:

1) Vous pouvez **créer une entrée numérique pour tester votre programme** qui a timing plus complexe que ce qui peut être produit en utilisant par l'un des ensembles de jeu de norme 'I/O' dispositifs, comme suit:

Définir une interruption fonction (appelons-le '**myIntr**') et faire '**attachInterrupt(0, myIntr, RISING)**' à l'intérieur de votre '**setup()**'. Connecter un **Pulser** dispositif à pin2 - maintenant '**myIntr()**' sera exécuter chaque fois qu'un **Pulser** front montant se produit. Votre '**myIntr()**' fonction peut être un algorithme que vous avez programmé (en utilisant le compteur global variables, et peut-être même '**random()**') pour produire un forme d'Onde de votre propre conception sur tout type de '**OUTPUT**' pin (disons que c'est pin 9). À présent **SAUT** pin 9 à votre 'Uno' ou 'Mega' désiré 'INPUT'pin à appliquer le numérique forme d'Onde généré à cette entrée pin (pour tester la réponse de votre programme à ce forme d'Onde particulier). . Vous pouvez générer une séquence d'impulsions, ou des caractères série, ou simplement des transitions de bord, de complexité arbitraire et à intervalles variables. Veuillez noter que si votre programme appelle '**micros()**' (ou appelle tout fonction qui en dépend), son '**return**' valeur **Sera augmenté** par le temps passé à l'intérieur de votre '**myIntr()**' fonction chaque fois que l'interruption se déclenche. Vous pouvez créer une rafale rapide d'arêtes synchronisées avec précision en utilisant des appels vers '**delayMicroseconds()**' de à l'intérieur '**myIntr()**' (peut-être pour générer un entier **octet** d'un transfert de débit en bauds élevé), ou simplement générer une transition par interruption (peut-être pour générer **un peu** d'un transfert à faible débit en bauds) avec le **Pulser** dispositif '**Period**' choisi en fonction de vos besoins timing (rappelez-vous que **Pulser** limite son minimum '**Period**' à 50 microsecondes).

2) Vous pouvez **expérimenter avec des bouclages de sous-systèmes**:

Par exemple, déconnectez votre 'SERIAL' 'I/O' dispositif TX '00' pin (modifiez-le en blanc), puis **SAUT** 'Uno' ou 'Mega' pin '01' Retour à 'Uno' ou 'Mega' pin '00' émuler un bouclage matériel de l'ATmega '**Serial**' sous-système. Maintenant, dans votre test programme, à l'intérieur '**setup()**' fait une **unique** '**Serial.print()**' d'un mot ou caractère, et à l'intérieur de votre '**loop()**' renvoyer tous les caractères reçus (quand '**Serial.available()**') en faisant un '**Serial.read()**' suivi d'un '**Serial.write()**' et observez ensuite ce qui se passe. Vous pourriez observer qu'un même '**SoftwareSerial**' retour en boucle **va échouer** (comme dans la vraie vie - le logiciel ne peut pas faire deux choses à la fois).

Vous pouvez aussi essayer **SPI** bouclage en utilisant un **SAUT** pour relier le pin 11 (MOSI) au pin 12 (MISO).





3) Vous pouvez **compter le nombre et / ou mesurer l'espacement de transitions de niveau spécifiques sur tout 'Uno' ou 'Mega' sortie pin X** qui se produisent à la suite d'un complexe Instruction ou bibliothèque Arduino fonction (à titre d'exemples: '**analogWrite()**', ou '**OneWire::reset()**', ou '**Servo::write()**'), comme suit:

SAUT pin X interrompre pin 2 et à l'intérieur de votre '**myIntr()**' utiliser un '**digitalRead()**' et un '**micros()**' appel, et comparez-les aux niveaux et aux temps enregistrés (à partir des interruptions précédentes). Vous pouvez modifier la sensibilité des bords pour la prochaine interruption, si nécessaire, en utilisant '**detachInterrupt()**' et '**attachInterrupt()**' de **à l'intérieur** votre '**myIntr()**'. Notez que vous ne pourrez pas suivre pin les transitions qui se produisent trop étroitement ensemble (plus proche que le total exécution temps de votre '**myIntr()**' fonction), tels que ceux qui se produisent avec des transferts I2C ou SPI, ou avec un débit en bauds élevé '**Serial**' transferts (Même si votre interruption fonction ne perturberait pas le timing inter-bord de ces transferts produits par le matériel). Notez également que les transferts par logiciel (comme '**OneWire::write()**' et '**SoftwareSerial::write()**') sont délibérément protégé contre les interruptions (par le code de leur bibliothèque désactivant temporairement toutes les interruptions afin d'éviter les perturbations de timing), vous ne pouvez donc pas effectuer de mesures à l'intérieur de ceux qui utilisent cette méthode.






Bien que vous puissiez à la place effectuer ces mêmes mesures d'espacement des bords **visuellement** dans un **Numérique formes d'onde** fenêtre, si vous êtes intéressé par l'espacement minimum ou maximum sur un grand nombre de transitions, ou par le comptage des transitions, utilisez cette option '**myIntr()**' -plus- **SAUT** la technique est plus pratique. Et tu peux mesurer, par exemple, les variations d'espacement des transitions produites par programme principal (en raison de l'effet de votre logiciel prenant différentes trajectoires exécution de différentes durées exécution), faire un genre de programme "profilage".

Les Menus









Fichier:

<u>Chargez INO ou PDE Prog (ctrl-L)</u> 	Permet à l'utilisateur de choisir un programme fichier ayant l'extension sélectionnée. Le programme reçoit immédiatement un Analyser
<u>Modifier/Examiner (ctrl-E)</u>	Ouvre le programme chargé pour affichage / édition.
<u>Enregistrer</u> 	Enregistrer le programme édité retourne au programme fichier d'origine.
<u>Enregistrer Sous</u>	Enregistrer le contenu du programme modifié sous un autre nom fichier.
<u>Suivant ('#include')</u> 	Avance le Volet de Code pour afficher le prochain ' #include ' fichier
<u>précédent</u> 	Renvoie le Volet de Code afficher le fichier précédent
<u>Quitter</u>	Quitte UnoArduSim après avoir rappelé à l'utilisateur de sauvegarder les fichier modifiés.

Trouver:

<u>Monter Pile d'Appels</u> 	Aller à la fonction précédente dans la pile des appels - le Volet de Variables s'adaptera à cette fonction
<u>Descendre Pile d'Appels</u> 	Aller à la prochaine fonction appelée dans la pile d'appels - le Volet de Variables s'adaptera à cette fonction
<u>Définir le texte Rechercher (ctrl-F)</u> 	Activer le Barre d'outils Trouver zone de saisie pour définir le texte à rechercher dans la suite (et ajoute le premier mot de la ligne en surbrillance dans la Volet de Code ou Volet de Variables si l'un de ceux a la mise au point).
<u>Trouver Texte suivant</u> 	Passer à la prochaine occurrence de texte dans le Volet de Code (si elle a le focus actif), ou à la prochaine occurrence de Texte dans Volet de Variables (si au lieu de cela il a le focus actif).
<u>Trouver Texte précédent</u> 	Aller à l'occurrence de texte précédente dans le Volet de Code (si elle a le focus actif), ou à la précédente occurrence de texte dans Volet de Variables (si au lieu de cela il a le focus actif).

Exécuter:

<u>Un Pas Dans (F4)</u>		Les étapes exécution sont transmises par une instruction ou <i>dans un appelé fonction</i> .
<u>Un Pas Franchir (F5)</u>		Les étapes exécution sont transmises par une instruction ou <i>par un appel fonction complet</i> .
<u>UnPas Sortir (F6)</u>		Avances exécution par <i>juste assez pour quitter le fonction actuel</i> .
<u>Exécuter Vers (F7)</u>		Exécute le programme, <i>arrêt à la ligne programme souhaitée</i> - vous devez d'abord cliquer sur surlinger pour sélectionner la ligne programme souhaitée avant d'utiliser Exécuter Vers.
<u>Exécuter Jusqu'à (F8)</u>		Exécute le programme jusqu'à ce qu'une écriture apparaisse sur le variable qui contenait le surlinger actuel dans Volet de Variables (cliquez sur l'un pour établir le surlinger initial).
<u>Exécuter (F9)</u>		Lance le programme.
<u>Arrêtez (F10)</u>		Arrête programme exécution (<i>et fige le temps</i>).
<u>Réinitialiser</u>		Réinitialise le programme (tous les variables de valeur sont réinitialisés à la valeur 0 et tous les pointeurs variables sont réinitialisés à 0x0000).
<u>Animer Exécution</u>		Trace automatiquement les lignes programme consécutives <i>avec délai artificiel ajouté</i> et mise en évidence de la ligne de code actuelle. Le fonctionnement en temps réel et les sons sont perdus.
<u>Ralenti</u>		Ralentit le temps d'un facteur 10.

Options:

<u>Un Pas Franchir Structors/Les opérateurs</u>	Parcourez les constructeurs, les destructeurs et la surcharge de l'opérateur fonctions lors de tout pas en avant (c'est-à-dire qu'il ne s'arrêtera pas dans ces fonctions).
<u>Registre-Allocation</u>	Affectez des sections fonction aux registres libre ATmega au lieu de la pile (génère une utilisation de mémoire vive légèrement réduite).
<u>Erreur sur non initialisé</u>	Indiquez comme une erreur Analyser n'importe où votre programme essaie d'utiliser un variable sans avoir au préalable initialisé sa valeur (ou au moins une valeur dans un tableau).
<u>Ajoutée 'loop()' Retard</u>	Ajoute 1000 microsecondes de retard à chaque fois 'loop()' est appelé (au cas où il n'y aurait pas d'autres appels programme à 'delay()' n'importe où) - utile pour éviter de prendre trop de retard sur le temps réel.
<u>Autoriser les interruptions imbriquées</u>	Autoriser la réactivation avec 'interrupts()' de l'intérieur d'une routine de service d'interruption utilisateur.

Configurer:

<u>'I/O' Dispositifs</u>	Ouvre une boîte de dialogue pour permettre à l'utilisateur de choisir le (s) type (s) et les numéros du 'I/O' dispositifs souhaité. Dans cette boîte de dialogue, vous pouvez également utiliser Enregistrer 'I/O' dispositifs en texte fichier et / ou Chargez 'I/O' dispositifs à partir d'un texte fichier précédemment enregistré (ou modifié) (y compris toutes les connexions pin, paramètres cliquables et valeurs saisies).
<u>Préférences</u>	Ouvre une boîte de dialogue permettant à l'utilisateur de définir des préférences, y compris l'indentation automatique des lignes source programme, autorisant la syntaxe Expert, le choix de la police police de caractères, optant pour une taille de police plus grande, respectant les limites tableau, autorisant les mots clés d'opérateur logique et affichant programme téléchargement. , choix de la version 'Uno' ou 'Mega' circuit imprimé et longueur du tampon TWI (pour I2C dispositifs).

VarRafraîchir:

<u>Autoriser Auto (-) Contract</u>	Autoriser UnoArduSim à contract afficher agrandie tableaux / objets en cas de retard.
<u>Minimal</u>	Seulement actualiser le Volet de Variables afficher 4 fois par seconde.
<u>Surlinger Changements</u>	Surlinger a modifié les valeurs de variable lors de l'exécution (peut provoquer un ralentissement).

Fenêtres:

<u>'Serial' Moniteur</u>	Connectez une entrée / sortie série dispositif à pins 0 et 1 (si aucune) et tirez-la vers le haut. 'Serial' moniteur TX / RX texte fenêtre.
<u>Tout restaurer</u>	Restaurez tous les enfants fenêtres minimisés.
<u>Pin Formes d'Ondes Numérique</u>	Restaurez un Pin Formes d'Ondes Numérique fenêtre réduit.
<u>Pin Forme d'Onde Analogique</u>	Restaurez un Pin Forme d'Onde Analogique fenêtre réduit.

Aide:

<u>Quick Aide Fichier</u>	Ouvre le fichier PDF UnoArduSim_QuickHelp fichier.
<u>Plein Aide Fichier</u>	Ouvre le fichier PDF UnoArduSim_FullHelp fichier.
<u>Bogue Corrections</u>	Afficher les correctifs importants pour bogue depuis la version précédente.
<u>Changement / Améliorations</u>	Affichez les modifications et améliorations significatives apportées depuis la version précédente.
<u>Sur</u>	Affiche la version, copyright.

'Uno' ou 'Mega' Circuit imprimé et 'I/O' Dispositifs

Le 'Uno' ou 'Mega' et le 'I/O' dispositifs sont tous modélisés électriquement avec précision, et vous pourrez avoir une bonne idée chez vous du comportement de votre programmes avec le matériel réel. Tous les systèmes électriques pin et conflits seront signalés.

Timing

UnoArduSim exécute suffisamment rapidement sur un PC ou une tablette pour qu'il le puisse (*dans la majorité des cas*) modéliser les actions programme en temps réel, **mais seulement si votre programme intègre** au moins un petit '`delay()`' appels ou autres appels qui le garderont naturellement synchronisé en temps réel (voir ci-dessous).

Pour ce faire, UnoArduSim utilise un temporisateur de rappel Fenêtres, fonction, qui lui permet de garder une trace précise du temps réel. Le exécution d'un certain nombre d'instructions programme est simulé au cours d'une tranche de la minuterie, et les instructions nécessitant une durée plus longue exécution (comme des appels à '`delay()`') peut avoir besoin d'utiliser plusieurs tranches de minuteur. Chaque itération de la minuterie de rappel fonction corrige l'heure système à l'aide de l'horloge matérielle de sorte que programme exécution soit constamment ajusté pour rester synchronisé avec le temps réel. *Les seuls temps exécution taux doit prendre du retard en temps réel* est quand l'utilisateur a créé des boucles serrées **sans délai supplémentaire** , ou 'I/O' dispositifs sont configurés pour fonctionner avec des fréquences 'I/O' dispositif (et / ou débit en bauds) très élevées, ce qui générerait un nombre excessif d'événements de modification de niveau pin et une surcharge de traitement associée. UnoArduSim gère cette surcharge en ignorant certains intervalles de la minuterie pour compenser, ce qui ralentit la progression de programme vers *ci-dessous en temps réel* .

En outre, programmes avec le grand tableaux affiché ou ayant à nouveau des boucles serrées **sans délai supplémentaire** peut causer une fréquence d'appel élevée au fonction et générer un **Volet de Variables** afficher la charge de mise à jour le retardant en temps réel - UnoArduSim réduit automatiquement la fréquence d'actualisation du variable pour essayer de suivre le rythme, mais si une réduction supplémentaire est nécessaire, choisissez **Minimal**, du **VarRafraîchir** menu pour ne spécifier que quatre actualisations par seconde.

Modéliser avec précision le temps exécution inférieur à la milliseconde pour chaque instruction ou opération programme **n'est pas fait** - seules des estimations très approximatives ont été adoptées pour la plupart à des fins de simulation. Cependant, le timing de '`delay()`' , et '`delayMicroseconds()`' fonctions et fonctions '`millis()`' et '`micros()`' sont tous parfaitement précis, et **tant que vous utilisez au moins un des délais fonctions** en boucle quelque part dans votre programme, **ou** vous utilisez un fonction qui se lie naturellement au fonctionnement en temps réel (comme '`print()`' qui est lié au débit en bauds choisi), puis les performances simulées de votre programme seront très proches du temps réel (encore une fois, sauf en cas d'événement de changement de fréquence extrêmement excessif et de niveau pin excessif ou de mises à jour excessives de Variables autorisées par l'utilisateur qui pourraient le ralentir).

Pour voir l'effet des instructions individuelles du programme *w poule en cours d'exécution* , il peut être souhaitable de pouvoir ralentir les choses. Un facteur de ralentissement temporel de 10 peut être défini par l'utilisateur dans le menu. **Exécuter** .

'I/O' Dispositif Timing

Ces dispositifs virtuels reçoivent une signalisation en temps réel des modifications qui se produisent sur leur entrée pins et produisent des sorties correspondantes sur leur sortie pins qui peuvent ensuite être détectées par le 'Uno' ou 'Mega'. Elles sont donc synchronisées de manière inhérente avec programme exécution. 'I/O' interne dispositif timing interne est défini par l'utilisateur (par exemple via la sélection débit en bauds ou la fréquence d'horloge), et des événements de simulateur sont configurés pour suivre le fonctionnement interne en temps réel.

Des sons

Chaque 'PIEZO' dispositif produit un son correspondant aux modifications du niveau électrique survenant sur le pin connecté, quelle que soit la source de ces modifications. Pour que les sons restent synchronisés sur le programme exécution, UnoArduSim démarre et arrête la lecture d'un tampon de son associé lorsque le exécution est démarré / arrêté.

Depuis la V2.0, le son a été modifié pour utiliser l'API audio Qt - malheureusement, son `QAudioOutput` 'class' ne prend pas en charge la mise en boucle du tampon sonore pour éviter de manquer d'échantillons sonores (comme cela peut arriver lors de longs délais d'exploitation de fenêtres OS). Par conséquent, afin d'éviter la grande majorité des clics sonores gênants et des ruptures de son lors des retards dans le système d'exploitation, le son est désormais mis en sourdine conformément à la règle suivante:

Le son est mis en sourdine aussi longtemps que UnoArduSim n'est pas la fenêtre "actif" (sauf lorsqu'un nouvel enfant fenêtre vient d'être créé et activé), **et même** Quand UnoArduSim est la fenêtre principale "actif" mais que le pointeur de la souris est **à l'extérieur** de son principal domaine client fenêtre.

Notez que cela implique que le son sera temporairement en sourdine en tant que roi alors que le pointeur de la souris plane **sur un enfant fenêtre**, et sera mis en sourdine **si on clique sur cet enfant fenêtre pour l'activer** (jusqu'à ce que l'on clique à nouveau sur le UnoArduSim fenêtre principale pour le réactiver) .

Vous pouvez toujours désactiver le son en cliquant n'importe où dans la zone cliente de la fenêtre principale d'UnoArduSim.

En raison de la mise en mémoire tampon, s'ensuit un décalage en temps réel pouvant aller jusqu'à 250 millisecondes par rapport à l'heure de l'événement correspondant sur le pin du 'PIEZO' attaché.

Limitations et éléments non pris en charge

Fichiers inclus

A '<>' - entre crochets '#include' de '<Servo.h>', '<Wire.h>', '<OneWire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' et '<SD.h>' est pris en charge, mais ceux-ci sont uniquement émulés - les fichiers réels ne sont pas recherchés; au lieu de cela, leurs fonctionnalités sont directement "intégrées à" UnoArduSim et sont valides pour la version corrigée d'Arduino.

Tout cité '#include' (par exemple de "supp.ino", "myutil.cpp", ou "mylib.h") est pris en charge, mais tous ces fichiers doivent **résider dans le même répertoire en tant que parent programme fichier** cette contient leur '#include' (il n'y a pas de recherche dans d'autres répertoires). le '#include' Cette fonctionnalité peut être utile pour minimiser la quantité de code programme indiquée dans le **Volet de Code** à n'importe quel moment. Header fichiers avec '#include' (c'est-à-dire ceux ayant un ".h" extension) fera en outre que le simulateur essaie d'inclure le fichier du même nom ayant un ".cpp" extension (si elle existe également dans le répertoire du parent programme).

Allocation dynamique de mémoire et RAM

Les opérateurs 'new' et 'delete' sont supportés, tout comme les Arduino natifs 'String' objets, **mais pas d'appels directs à 'malloc()', 'realloc()' et 'free()'** que ceux-ci s'appuient.

Une utilisation excessive de la RAM pour les déclarations variable est signalée à l'heure Analyser et le dépassement de capacité de la mémoire RAM est signalé pendant programme exécution. Un article au menu **Options** vous permet d'émuler l'allocation de registre ATmega normale, comme le ferait l'AVR Acompilateur, ou de modéliser un schéma de compilation alternatif utilisant uniquement la pile (en tant qu'option de sécurité dans le cas où un bogue apparaît dans ma modélisation d'allocation de registre). Si vous utilisez un pointeur pour examiner le contenu de la pile, il doit refléter avec précision ce qui apparaîtrait dans une implémentation matérielle réelle.

Allocations de mémoire 'Flash'

'Flash' mémoire 'byte', 'int' et 'float' variables / tableaux et leur accès en lecture fonctions correspondant sont pris en charge. Tout 'F()' Appel fonction ('Flash'-macro) de toute chaîne littérale est prise en charge, mais la seule chaîne prise en charge fonctions à accès direct 'Flash' est 'strcpy_P()' et 'memcpy_P()', donc, pour utiliser un autre fonctions, vous devez d'abord copier la chaîne 'Flash' dans une RAM normale. 'String' variable, et ensuite travailler avec cette RAM 'String'. Lorsque vous utilisez le 'PROGMEM' variable-mot-clé modificateur, il doit apparaître **en face de** le nom variable, et que variable **doit également être déclaré** comme 'const'.

'String' Variables

Le natif 'String' La bibliothèque est presque complètement supportée avec quelques exceptions très (et mineures).

le 'String' opérateurs pris en charge sont +, + =, <, <=, >, > =, ==, !=, et []. Notez que: 'concat()' prend un **unique** argument qui est le 'String', ou 'char', ou 'int' être annexé à l'original 'String' objet, **ne pas** deux arguments comme indiqué à tort sur les pages Web de référence Arduino).

Bibliothèques Arduino

Seulement 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h', 'OneWire.h', 'Stepper.h', 'SD.h', 'TFT.h' et 'EEPROM.h' pour le **V1.8.8 Arduino** RELEASE sont actuellement pris en charge dans UnoArduSim. V2.6 introduit un mécanisme pour 3^e partie de support de bibliothèque par l'intermédiaire d' fichiers prévu dans la 'include_3rdParty' dossier qui se trouvent dans le répertoire d'installation UnoArduSim. Essayer '#include' les ".cpp" et ".h" fichiers d'autres non encore non pris en charge Les bibliothèques ***travail non*** car ils contiendront des instructions d'assemblage de bas niveau et directives non pris en charge et fichiers non reconnu!

Pointeurs

Les pointeurs vers les types simples, tableaux ou objets sont tous pris en charge. Un pointeur peut être assimilé à un tableau du même type (par exemple 'iptr = intarray'), mais alors il y aurait *pas de vérification ultérieure des limites tableaux* sur une expression comme 'iptr[index]'.

Fonctions peut renvoyer des pointeurs, ou 'const' pointeurs, mais tout niveau ultérieur de 'const' sur le pointeur renvoyé est ignoré.

Il y a ***pas de support*** pour les appels fonction effectués via ***fonction-pointeurs déclarés par l'utilisateur***.

'class' et 'struct' Objets

Bien que le poly-morphisme et l'héritage (quelle que soit la profondeur) soient pris en charge, une 'class' ou 'struct' ne peut être défini pour avoir au plus **un** base 'class' (c'est à dire **plusieurs**- l'héritage n'est pas supporté). Les appels d'initialisation du constructeur de base 'class' (via la notation entre deux points) dans les lignes de déclaration du constructeur sont pris en charge, mais **ne pas** membre-initialisations en utilisant cette même notation du colon. Cela signifie que objets qui contient 'const' les variables non 'static' ou les types variables de référence ne sont pas pris en charge (ceux-ci ne sont possibles qu'avec des initialisations de membre au moment de la construction spécifiées)

Les surcharges d'opérateur d'assignation de copie sont prises en charge avec les constructeurs et les assignations de déplacement, mais la conversion objet définie par l'utilisateur ("type-casté") fonctions n'est pas prise en charge.

Échelle

Il n'y a pas de soutien pour le 'using' mot-clé, ou pour 'namespace', ou pour 'file' échelle. Toutes les déclarations non locales sont supposées globales par implémentation.

Tout 'typedef', 'struct', ou 'class' définition (c'est-à-dire pouvant être utilisé pour de futures déclarations), doit être faite **global** échelle (**local** les définitions de tels éléments à l'intérieur d'un fonction ne sont pas prises en charge).

Qualificatifs 'unsigned', 'const', 'volatile', 'static'

le 'unsigned' prefix fonctionne dans tous les contextes juridiques normaux. le 'const' le mot-clé, lorsqu'il est utilisé, doit **précéder** le nom variable ou le nom fonction ou 'typedef' name qui est déclaré - le placer après le nom provoquera une erreur Analyser. Pour Déclarations fonction, seul le fonctions renvoyant le pointeur peut avoir 'const' apparaissent dans leur déclaration.

Tout UnoArduSim variables sont '**volatile**' par mise en œuvre, donc le '**volatile**' le mot clé est simplement ignoré dans toutes les déclarations variable. Fonctions ne sont pas autorisés à être déclarés '**volatile**' , ni les arguments d'appel fonction.

le '**static**' mot clé est autorisé pour variables normal et pour les membres objet et membre-fonctions, mais est explicitement interdit pour les instances objet elles-mêmes ('**class**' / '**struct**'), pour fonctions non membre et pour tous les arguments fonction.

Directives Acompilateur

'**#include**' et régulier '**#define**' sont tous deux pris en charge, mais ***pas de macro*** '**#define**'. le '**#pragma**' directive et directives d'inclusion conditionnelle ('**#ifndef**' , '**#ifndef**' , '**#if**' , '**#endif**' , '**#else**' et '**#elif**') sont également ***non supporté***. le '**#line**' , '**#error**' macros prédéfinies (comme '**_LINE_**' , '**_FILE_**' , '**_DATE_**' , et '**_TIME_**') sont également ***non supporté***.

Éléments de langue Arduino

Tous les éléments de la langue Arduino sont pris en charge, à l'exception de la version douteuse. '**goto**' les instructions (la seule utilisation raisonnable que je puisse imaginer serait comme un saut (boucle sans fin et arrêt sécurisé) en cas d'erreur que votre programme ne pourrait pas traiter autrement)

C / C ++ - éléments de langage

Les "qualificatifs de champ binaire" permettant d'économiser des bits pour les membres dans les définitions de structure sont: ***non supporté***.

'**union**' est ***non supporté***.

L'opérateur étrange "comma" est ***non supporté*** (vous ne pouvez donc pas utiliser plusieurs expressions séparées par des virgules lorsqu'une seule expression est normalement attendue, par exemple dans '**while()**' et '**for(; ;)**' constructions).

Fonction Modèles

fonctions défini par l'utilisateur qui utilise le mot clé "template" pour lui permettre d'accepter des arguments de type "générique" ***non supporté***.

Emulation en temps réel

Comme indiqué ci-dessus, exécution fois parmi les nombreuses instructions individuelles possibles pour Arduino programme sont ***ne pas*** modélisé avec précision, de sorte que pour fonctionner à une vitesse en temps réel, votre programme aura besoin d'une sorte de '**delay()**' instruction (au moins une fois par '**loop()**'), ou une instruction naturellement synchronisée avec les modifications en temps réel du niveau pin (telles que '**pulseIn()**' , '**shiftIn()**' , '**Serial.read()**' , '**Serial.print()**' , '**Serial.flush()**' etc.).

Voir **Timing** et **Des sons** ci-dessus pour plus de détails sur les limitations.

Notes de version

Bogue Corrections

V2.8.1- Juin 2020

- 1) Les lignes vierges supplémentaires supprimées par la préférence de mise en forme automatique ont entraîné un décalage vers le bas de la ligne initialement mise en surbrillance dans Modifier/Examiner par le nombre de lignes vides qui ont été supprimées au-dessus.
- 2) 'analogRead()' n'acceptait que le numéro complet numérique pin.
- 3) Un 'class' qui contenait des surcharges fonction ne différant que par l'attribut 'unsigned' d'un argument fonction pourrait avoir la mauvaise surcharge fonction appelée. Cela a affecté LCD 'print(char/int/long)' où la surcharge d'impression LK4 base-10 fonction serait appelée à la place, et cela a conduit LCD 'printFloat()' à imprimer '46' au lieu du point décimal '.'.
- 4) L'insertion automatique (sur 'Enter') d'un intégré déjà apparié n'a pas fonctionné après le retour en arrière pour corriger une erreur de frappe sur la ligne.
- 5) Le dispositif 'TFT' avec un 'RS' vierge pourrait provoquer un plantage lors de l'exécution.

V2.8.0- Juin 2020

- 1) Lorsqu'un avertissement d'Analyser (mais pas une erreur d'Analyser) se produisait, la V2.7 pouvait tenter de mettre en surbrillance la ligne problématique dans le mauvais tampon (dans le tampon '**#include**' le plus récent à la place), et cela provoquerait un crash silencieux (même avant le pop-up d'avertissement apparaît) si le numéro de ligne était au-delà de la fin de ce tampon '**#include**'.
- 2) Les octets reçus dans les fenêtres de moniteur plus grandes des appareils 'I2CSLV', 'SPISLV', 'TFT', 'LCDI2C' et 'LDCSPI' n'étaient toujours pas tous signalés correctement (cela n'affectait pas leur fonctionnement).
- 3) Variables de type '**unsigned char**' ont été promus à taper '**int**' dans les expressions arithmétiques, mais a incorrectement maintenu leur '**unsigned**' état, conduisant à un défaut '**unsigned**' expression résultante.
- 4) La modification (ou la suppression) du pin d'un 'LED4' ou '7SEG' dispositif à partir d'un paramètre pin initial valide n'a pas réussi à détacher ses 3 pins supérieurs et pourrait entraîner des plantages inexplicables - en outre, les modifications apportées à la V2.7 pour harmoniser la gestion de tous les 'LED' complètement cassé le 'LED4' dispositif.
- 5) Une erreur dans les modifications apportées à la version 2.6 pour prendre en charge les interruptions 'LOW' a provoqué des interruptions 'FALLING' attachées à pin 3 pour corrompre la détection de changement de niveau d'interruption sur pin 2.
- 6) 'SoftwareSerial' désactivait de manière incorrecte les interruptions utilisateur pendant chaque caractère qu'il recevait.
- 7) Lorsqu'un programme 'PROGIO' contenant une erreur d'analyse a été chargé, l'erreur a été signalée mais ce programme avait déjà été remplacé par un programme 'PROGIO' par défaut dans la fenêtre du moniteur 'PROGIO'.
- 8) Depuis la version V2.4, les modifications pin sur Pin 1 et Pin 0 n'ont pas été vues pendant le téléchargement.
- 9) Les opérations de lecture ou d'écriture en boucle sur un SD fichier ouvert ont provoqué l'échec du séquençement exécution, entraînant une éventuelle erreur interne UnoArduSim en raison de la profondeur de niveau échelle.
- 10) Tenter de changer le 'MISO' pin sur un 'SD_DRV' dispositif a corrompu le MOSI pin.
- 11) Toutes les versions précédentes n'ont pas averti que l'utilisation '**analogWrite()**' sur pins 9 ou 10 corromprait 'Servo' timing pour tous les 'Servo' dispositifs actifs.

V2.7.0- Mars 2020

- 1) Lorsque le thème du système d'exploitation léger (Fenêtres par défaut) a été adoptée, la **Volet de Code** ne montrait pas la couleur mettant en évidence introduite dans V2.6 (au lieu, seul un hexadécimal gris résultant d'une substitution du système).
- 2) Version 2.6 auto-indent-onglet par inadvertance cassé le formatage à la première `'switch()'` construction.
- 3) La nouvelle fonctionnalité de navigation appel pile introduit dans la version 2.6 a montré **des valeurs incorrectes** pour locale variable lorsqu'ils ne sont pas à l'intérieur du libre en cours d'exécution, et a échoué avec imbrications membres libre.
- 4) La version 2.6 a introduit une bogue qui affiche la valeur incorrecte pour 'RX' présent et passé octets pour 'I2CSLV', 'SPISLV', dispositif 'TFT', 'LCDI2C' and 'LCDSPI'(et leur moniteur fenêtre).
- 5) Une modification apportée à V2.4 a provoqué Exécuter | Animer Exécution mettant en lumière de sauter sur plusieurs lignes de code exécute.
- 6) Depuis V2.4, de assertive 'SS*' ou 'CS*' sur un 'I/O' contract dans l'instruction immédiatement après une `'SPI.transfer()'` causerait que contract à ne pas obtenir de l'octet de données transférées. En outre, l'octet réception `'SPI_MODE1'` et `'SPI_MODE3'` n'a pas été signalé avant le début de l'octet suivant envoyé par le maître (et l'octet a été complètement perdue si on désélectionné la contract 'CS*' avant cette date).
- 7) Dans la nouvelle `'SPI_SLV'` le mode autorisé depuis V2.4, `'bval = SPI.transfer()'` seulement retourné la valeur correcte `'bval'` si le transfert d'octet était déjà complet et attendre quand `'transfer()'` a été appelé.
- 8) La zone d'édition 'DATA' sur 'SPISLV' dispositif obtient maintenant la valeur par défaut 0xFF quand il n'y a plus d'octets à répondre avec.
- 9) La synchronisation état DEL était incorrecte pour 'PSTEPR' dispositif ayant plus d'une micro-étape par étape complet.
- 10) conflit électrique causée par 'I/O' dispositif réagir à des transitions de l'horloge 'SPI', un signal de 'PWM', ou `'tone'` signal n'a pas été rapporté, et pourrait conduire à inexplicable (corrompu) réception de données.
- 11) Lorsque l'intervalle entre les interruptions était trop faible (inférieure à 250 microsecondes), un (défectueux) changement de V2.4 modifié le échelle fonction de intégré que Les compteurs d'utiliser les deux systèmes, ou des boucles d'instruction, pour générer des retards (exemples de chacun sont `'delay()'` et `'delayMicroseconds()'`). Une modification ultérieure de V2.5 causé défaut d'alignement de `'shiftOut()'` des données et des signaux d'horloge lorsqu'une interruption est passé entre les bits.
- 12) Accepter un texte automatique d'achèvement intégré libre via la touche Entrée n'a pas réussi à dépouiller les types de paramètres du texte de l'appel libre inséré.
- 13) Chargement d'un nouveau (utilisateur-interruption conduit) pins lorsqu'un avait encore pins en cours d'exécution précédemment une interruption en attente pourrait causer un accident pendant le téléchargement (en raison de la tentative défectueuse exécuté de la nouvelle routine d'interruption).
- 14) Surligner membres auto-complétion (accessible via 'ALT'-flèche vers la droite) pour objet intérieur `'#include'` fichier sont désormais accessibles dès leur `'#include'` agrandie est avec succès analyser.
- 15) Une déclaration libre ayant un espace intérieur par inadvertance un nom de paramètre libre a provoqué un message d'erreur Analyser pas claire.
- 16) Lorsque exécuté arrêté dans un autre module de la principale pins, le Fichier | l'action précédente a échoué à devenir activée.
- 17) accolades cité unique (`'{'` et `'}'`) Étaient encore comptées (à tort) comme supports dans la prototype Analyser, et aussi confus formatage-indent auto-onglet ..
- 18) `'OneWire::readBytes(byte* buf, int count)'` avait été immédiatement mise à jour parviennent pas à l'affiche `'buf'` contenu dans le Volet de Variables.
- 19) Octal-bascule de sortie 'OWISLV' dispositif analyseur a montré que les niveaux retardé d'une écriture de verrouillage registre.

V2.6.0- Jan 2020

- 1) Un bogue introduit dans V2.3 conduit à un accident lorsque la valeur ajoutée **Fermer** bouton a été utilisé dans la **Trouver / Remplacer** dialogue (plutôt que son Quitter bouton barre de titre).
- 2) Si un utilisateur a pins '**#include**' d'autre part, l'utilisateur fichier **Enregistrer** bouton à l'intérieur **Modifier/Examiner** aurait pas fait sauver un agrandie modifié s'il y avait une erreur Analyser ou Exécuté existant marqué dans un autre agrandie.
- 3) UNE **Annuler** après un **Enregistrer** pourrait également être source de confusion - pour ces raisons, le **Enregistrer** et **Annuler** fonctionnalités de bouton ont été modifiés (voir **Les changements et améliorations**).
- 4) '**TFT :: text ()**' fonctionnait, mais les fonctions '**TFT :: print ()**' ne l'étaient pas (elles étaient simplement bloquées pour toujours). De plus, '**TFT :: loadImage ()**' a échoué si '**Serial.begin ()**' avait été fait plus tôt (ce qui est le cas normal et est maintenant requis).
- 5) Supports déséquilibrées dans une définition de 'class' pourrait provoquer un blocage depuis V2.5.
- 6) Essai direct logique sur les valeurs '**long**' est revenu '**false**' si aucun des 16 bits les plus bas ont été établis.
- 7) UnoArduSim avait été une erreur en berne lorsqu'un pointeur variables a été déclarée comme la boucle variables à l'intérieur des crochets d'une instruction '**for ()**'.
- 8) UnoArduSim avait été désaveu tests logiques comparant des pointeurs vers '**NULL**' ou '**0**'.
- 9) UnoArduSim avait été désaveu arithmétique des pointeurs impliquant un nombre entier variables (seul entier constantes ont été autorisées).
- 10) Interruptions définies à l'aide '**attachInterrupt (pin, name_func, LOW)**' ont été détectés seulement sur une transition vers '**LOW**'.
- 11) Lors de l'utilisation de plusieurs périphériques 'I2CSLV', un esclave non adressé peut interpréter les données de bus ultérieures comme correspondant à son adresse de bus (ou appel global 0x00), et ainsi faussement signaler ACK, corrompre le niveau ACK de bus et suspendre une '**requêteFrom ()**'.
- 12) En passant la valeur numérique '**0**' (ou '**NULL**') comme argument fonction à un pointeur dans un appel fonction est maintenant autorisé.
- 13) Le niveau de retrait après une imbrication de tabulation des constructions de '**switch ()**' était trop faible lorsque le choix de 'auto-indent formatting' de **Configurer | Préférences** a été utilisé.
- 14) Soustraction de deux pointeurs compatibles entraîne maintenant un type de '**int**'.
- 15) UnoArduSim avait attendu un constructeur par défaut défini par l'utilisateur pour un membre objet même si elle n'a pas été déclarée comme '**const**'.
- 16) Lors d'une pause exécution, la position tirée d'un 'STEPR', 'SERVO' ou 'MOTOR' moteur pourrait par décalage jusqu'à 30 millisecondes de mouvement derrière sa dernière position calculée réelle.
- 17) '**Stepper :: setSpeed (0)**' a été à l'origine d'un accident à cause d'une division par zéro.
- 18) '**if ()**' unique en ligne, '**for ()**' et '**else**' construit ne provoquent plus un trop d'onglets indentation automatique.

V2.5.0- octobre 2019

- 1) A bogue introduit dans V2.4 cassé l'initialisation de la carte 'SD' (causé un accident).
- 2) En utilisant le sous-système de 'SPI' dans le nouveau '**SPI_SLV**' Mode a travaillé de manière incorrecte dans '**SPI_MODE1**' et '**SPI_MODE3**'.
- 3) Auto-complétion des pop-ups (comme demandé par 'ALT-right=arrow') ont été fixés pour fonctions objet ayant paramètres; la liste pop-ups incluent maintenant les membres de la classe héritée (base-) et complétions automatiques apparaissent maintenant aussi '**Serial**'.
- 4) Depuis V2.4, la valeur de retour pour '**SPI.transfer ()**' et '**SPI.transfer16 ()**' était incorrecte si une routine d'interruption de l'utilisateur a tiré lors de ce transfert.

- 5) Dans V2.4, rapide des formes d'ondes périodiques ont été présentés comme ayant une plus longue durée que leur durée réelle évidente lorsqu'elle est vue à zoom supérieur.
- 6) le constructeur `'File::File(SdFile &sdf, char *fname)'` ne fonctionnait pas, donc `'File::openNextFile()'` (Qui repose sur ce constructeur) ne fonctionnait.
- 7) UnoArduSim a été une erreur mauvaise déclaration Analyser sur objet-variable et objet-retour fonctions, a déclaré que 'static'.
- 8) déclarations d'affectation avec un 'Servo', 'Stepper', ou 'OneWire' objet variables sur le LHS, Et un objet-retour fonction ou constructeur sur l'ERS, a provoqué une erreur de calcul interne du nombre de objets associé, conduisant à une éventuelle crash.
- 9) des tests booléennes sur objets de type 'File' étaient toujours retour 'true' même si le fichier n'a pas été ouverte.

V2.4– mai 2019

- 1) Les points de contrôle Exécuter Jusqu'à pourraient être faussement déclenchés par une écriture dans un variable adjacent (adresse inférieure de 1 octet).
- 2) Des sélections de débit en bauds pourraient être détectées quand la souris était à l'intérieur d'un 'SERIAL' ou 'SFTSER' dispositif Baud liste déroulante (même quand aucun débit en baud n'a été cliqué).
- 3) Depuis la V2.2, des erreurs de réception en série se sont produites à un débit de 38400 bauds.
- 4) Un changement effectué dans la V2.3 a causé 'SoftwareSerial' signaler parfois de manière erronée une interruption désactivée (et échouera dès la première réception RX).
- 5) Une modification apportée à la version V2.3 a amené SPISLV dispositifs à interpréter de manière erronée les valeurs hexadécimales entrées directement dans leur zone de saisie 'DATA'.
- 6) Une modification apportée à la V2.3 a provoqué SRSLV dispositifs parfois, à tort et à tort, détecter un conflit électrique sur leur 'Dout' pin et ainsi interdire une affectation du pin dans ce dernier. Des tentatives répétées pour attacher un pin à 'Dout' pourraient alors entraîner un crash éventuel une fois le dispositif retiré.
- 7) Tenter de rattacher un 'LED4' dispositif au-delà de pin 16 provoquerait un crash.
- 8) Un bogue déclenché par des appels répétés rapides à `'analogWrite(255)'` pour 'MOTOR' contrôle dans l'utilisateur programme a eu pour résultat 'MOTOR' les vitesses sont incorrectes (beaucoup trop lentes).
- 9) Depuis la V2.3, une 'Dout' pin n'a pas pu être affectée à SRSLV dispositifs en raison d'une détection électrique conflit défectueuse (et par conséquent, une affectation pin n'est pas autorisée).
- 10) Les esclaves SPI réinitialisent désormais leurs logiques d'émetteur et de récepteur lorsque leur 'SS*' pin va 'HIGH'.
- 11) Appel 'Wire.h' fonctions `'endTransmission()'` ou `'requestFrom()'` quand les interruptions sont actuellement \ désactivées génère maintenant une erreur exécution ('Wire.h' nécessite des interruptions activées pour fonctionner).
- 12) 'Ctrl-Home' et 'Ctrl-End' fonctionnent maintenant comme prévu dans Modifier/Examiner.
- 13) le 'OneWire' commande de bus 0x33 ('ROM_READ') ne fonctionnait pas et a accroché le bus.

V2.3– décembre 2018

- 1) Un bogue introduit dans la V2.2 rendait impossible la modification de la valeur d'un élément tableau dans **Modifier/Surveiller Variable**.
- 2) Depuis la version 2.0, le texte de la 'RAM free' Le contrôle de la barre d'outils n'était visible que si vous utilisiez un thème sombre Fenêtres-OS.
- 3) Sur **Fichier | Chargez** et sur les E / S dispositif fichier **Chargez**, les filtres fichier (comme '*.ino' et '*.txt') ne fonctionnaient pas - fichiers de tous les types ont été montrés à la place.
- 4) L'état "modifié" d'un fichier a été perdu après avoir **Accepter** ou **Compiler** dans une prochaine **Fichier | Modifier/Examiner si aucune autre modification n'a été faite** (**Enregistrer** est devenu invalide, et il n'y avait pas d'invite automatique à **Enregistrer** sur programme **Quitter**).

- 5) Les opérateurs `'/='` et `'%='` seulement donné des résultats corrects pour `'unsigned'` variables à gauche
- 6) Le conditionnel ternaire `'(bval) ? S1:S2'` a donné un résultat incorrect quand `'S1'` ou `'S2'` était une expression locale au lieu d'un variable.
- 7) Fonction `'noTone()'` a été corrigé pour devenir `'noTone(uint8_t pin)'`.
- 8) Un bogue de longue date a provoqué un accident après avoir quitté une **Réinitialiser** lorsque cette Réinitialiser a été effectuée au milieu d'une fonction appelée avec l'un de ses paramètres manquant (et recevant ainsi une valeur d'initialisation par défaut).
- 9) Expressions membres (par exemple `'myobj.var'` ou `'myobj.func()'`) n'héritaient pas de la `'unsigned'` propriété de leur côté droit (`'var'` ou `'func()'`) et ne pouvaient donc être directement comparés ni combinés avec d'autres `'unsigned'` types - une affectation intermédiaire à un `'unsigned'` variable a d'abord été requis.
- 10) UnoArduSim insistait sur le fait que si une définition de fonction; s avait un paramètre quelconque avec un initialiseur par défaut, que le fonction ait déclaré un prototype antérieur.
- 11) Appels à `'print(byte bvar, base)'` promu par erreur `'bvar'` à un `'unsigned long'`, et ainsi imprimé trop de chiffres.
- 12) `'String(unsigned var)'` et `'concat(unsigned var)'` et opérateurs `'+=(unsigned)'` et `'+(unsigned)'` créé de manière incorrecte `'signed'` des chaînes à la place.
- 13) Un 'R=1K' dispositif chargé depuis un **IODevices.txt** fichier avec position `'U'` a été dessiné par erreur avec son curseur (toujours) dans le **position opposée** de sa vraie position électrique.
- 14) Tenter de s'appuyer sur la valeur par défaut `'inverted=false'` argument lors de la déclaration d'un `'SoftwareSerial()'` objet a provoqué un crash, et en passant `'inverted=true'` ne fonctionnait que si l'utilisateur programme faisait une autre `'digitalWrite(txpin, LOW)'` afin d'établir d'abord le ralenti requis `'LOW'` niveau sur le **TX** pin.
- 15) 'I2CSLV' dispositifs n'a pas répondu aux modifications apportées aux zones d'édition pin (les valeurs par défaut A4 et A5 sont restées en vigueur).
- 16) 'I2CSLV' et 'SPISLV' dispositifs n'ont pas détecté et corrigé les éditions partielles lorsque la souris a quitté leurs bordures
- 17) Les valeurs Pin pour dispositifs qui suivaient un SPISLV ou un SRSLV n'étaient pas correctement enregistrées dans le fichier. **IODevs.txt** fichier en tant qu'hexadécimaux.
- 18) La tentative de connexion de plusieurs SPISLV dispositif MISO à pin 12 a toujours généré une erreur électrique Conflit.
- 19) Changer un pin de `'OUTPUT'` retour à **Mode** `'INPUT'` impossible de réinitialiser le niveau de verrouillage des données pin sur `'LOW'`.
- 20) En utilisant `'sendStop=false'` dans les appels à `'Wire.endTransmission()'` ou `'Wire.requestFrom()'` causé un échec.
- 21) UnoArduSim a incorrectement autorisé une `'SoftwareSerial'` réception se produire simultanément avec un `'SoftwareSerial'` transmission.
- 22) Variables déclaré avec `'enum'` Une nouvelle valeur après la ligne de déclaration n'a pas pu être affectée au type et UnoArduSim ne reconnaissait pas les membres 'enum' lorsqu'il était associé à un (légal) `'enumname::'` préfixe.

V2.2- juin. 2018

- 1) L'appel d'un fonction avec moins d'arguments que requis par sa définition (lorsque ce fonction était "défini en aval", c'est-à-dire lorsqu'il ne possédait aucune ligne de déclaration prototype antérieure) a provoqué une violation de mémoire et un blocage.
- 2) Depuis la V2.1, **Formes d'onde** n'avait pas été mis à jour pendant **Exécuter** (seulement à **Arrêtez** ou après un **Un Pas**) - en outre, **Volet de Variables** les valeurs ne sont pas mises à jour pendant une longue période **Un Pas** opérations.
- 3) Un peu mineur **Forme d'Onde** Les problèmes de défilement et de zoom qui existaient depuis la V2.0 ont maintenant été corrigés.

- 4) Même dans les premières versions, le Réinitialiser à $t = 0$ avec un PULSER ou un FUNCEN, dont la période ferait son dernier cycle avant $t = 0$ être seulement un **partiel** cycle, a eu pour résultat **Forme d'Onde** après $t = 0$ étant décalé de sa position réelle de cette quantité (ou de la fraction restante) du cycle, respectivement vers la droite ou vers la gauche.
- 5) Correction de quelques problèmes avec la coloration syntax-color dans **Modifier/Examiner**.
- 5) Depuis la V2.0, un clic sur agrandir, un objet dans un tableau de objets ne fonctionnait pas correctement.
- 6) '**delay(long)**' a été corrigé pour être '**delay(unsigned long)**' et '**delayMicroseconds(long)**' a été corrigé pour être '**delayMicroseconds(unsigned int)**'.
- 7) À partir de la V2.0, fonctions a été connecté à l'aide de '**attachInterrupt()**' ne sont pas contrôlés comme étant valides fonctions à cette fin (c'est-à-dire '**void**' retour, et n'ayant pas de paramètres d'appel).
- 8) L'impact de '**noInterrupts()**' sur fonctions '**micros()**', '**mills()**', '**delay()**', '**pulseInLong()**'; son impact sur '**Stepper::step()**' et sur '**read()**' et '**peek()**' délais d'attente; sur toute la réception série RX, et sur '**Serial**' transmission, est maintenant reproduite avec précision.
- 9) Le temps passé dans les routines d'interruption utilisateur est désormais pris en compte dans la valeur renvoyée par '**pulseIn()**', le retard produit par '**delayMicroseconds()**', et dans la position des arêtes affichées **Pin Formes d'Ondes Numérique**.
- 10) Appels à **objet-membre** fonctions qui faisaient partie d'expressions complexes plus grandes, ou étaient eux-mêmes à l'intérieur d'appels fonction ayant plusieurs arguments complexes, e, g, '**myobj.memberfunc1() + count/2**' ou '**myfunc(myobj.func1(), count/3)**', aurait des valeurs incorrectes calculées / passées à l'heure exécution en raison d'allocations d'espace de pile erronées.
- 11) Tableaux du pointeur variables fonctionnait correctement, mais les valeurs affichées erronées étaient affichées dans **Volet de Variables**.
- 12) Lorsque des tableaux dynamiques de type simple ont été créés avec '**new**', seul le premier élément obtenait une initialisation (par défaut) à la valeur 0 - maintenant, tous les éléments le font.
- 13) '**noTone()**', ou la fin d'un ton fini, ne réinitialise plus le pin (il reste '**OUTPUT**' et va '**LOW**').
- 14) La rotation continue du 'SERVO' dispositifs est désormais parfaitement stationnaire avec une largeur d'impulsion de 1 500 microsecondes.
- 15) L'appel sur **SdFile::ls()** (la liste du répertoire de la carte SD) a fonctionné correctement, mais n'a pas correctement montré des transferts SPI de blocs dupliqués dans le Waveforms fenêtre.

V2.1.1- mars 2018

- 1) Incohérences corrigées dans les paramètres régionaux non anglais avec la langue enregistrée dans '**myArduPrefs.txt**', avec les boutons de langue de radio affichés dans la **Préférences** boîte de dialogue, et avec correspondance aux lignes traduites dans '**myArduPrefs.txt**'.
- 2) Allocations avec '**new**' accepte maintenant une dimension tableau entière qui n'est pas une constante.
- 3) En cliquant dans le **Volet de Variables** agrandir un tableau multi-dimensionnel montrerait un vide superflu '**[]**' paire de support.
- 4) Références Tableau-element avec des caractères superflus à la fin (par exemple '**y[2]12**') n'ont pas été détectés comme des erreurs à l'heure de Analyser (les caractères supplémentaires étaient simplement ignorés).

V2.1- mars 2018

- 1) Un fichier bogue dans les nouvelles versions V2.0.x a entraîné une croissance du segment de mémoire Fenêtres à chaque mise à jour du fichier. **Volet de Variables** -- après des millions de mises à jour (plusieurs minutes de exécution), un crash pourrait en résulter.
- 2) Appels au 'static' membre fonctions utilisant le double-colon '**::**' la notation a échoué à Analyser quand à l'intérieur '**if()**', '**while()**', '**for()**', et '**switch()**' entre crochets et lorsque des expressions internes sont utilisées comme arguments d'appel fonction ou comme indices tableau.

V2.0.2 février 2018

- 1) Un bogue introduit dans la V2.0 a provoqué une **Fichier | Chargez** crash si un '**#include**' fait référence à un fichier manquant ou vide
- 2) À l'intérieur d'un **IOdevs.txt** fichier, il '**I/O**' le nom 'One-Shot' était attendu à la place de l'ancien 'Oneshot'; les deux sont maintenant acceptés.

V2.0.1– janvier 2018

- 3) En d'autres langues que l'anglais, '**en**' a été incorrectement indiqué comme sélectionné dans **Préférences** rendant maladroit le retour en anglais (nécessitant la désélection puis la re-sélection).
- 4) L'utilisateur avait pu laisser une valeur de zone d'édition Dispositif pin dans un état incomplet (comme 'A_') et laisser les bits 'DATA' d'un 'SRS:V' incomplets.
- 5) Le nombre maximum de curseurs Analogique avait été limité à 4 (corrigé à présent de 6).
- 6) UnoArduSim n'insiste plus sur '**=**' apparaissant dans une initialisation d'agrégat tableau.
- 7) UnoArduSim avait insisté pour que l'argument "inverted_logic" soit fourni à '**SoftwareSerial()**'.
- 8) Les opérations de décalage de bits permettent maintenant des décalages plus longs que la taille du variable décalé.

V2.0– Déc. 2017

- 1) Tous les fonctions déclarés comme '**unsigned**' avait néanmoins renvoyé des valeurs comme si elles étaient '**signed**'. Cela n'a aucun effet si le '**return**' valeur a été assigné à un '**unsigned**' variable, mais aurait provoqué un abus interprétation négative si **MSB == 1**, et il a ensuite été affecté à un '**signed**' variable, ou testé dans une inégalité.
- 2) Analogique Sliders atteignaient seulement un maximum '**analogRead()**' valeur de 1022, pas la valeur correcte 1023.
- 3) Un bogue introduit par inadvertance dans la V1.7.0 dans une logique utilisée pour accélérer le traitement du système SPI SCK pin a entraîné des transferts SPI pour '**SPI_MODE1**' et '**SPI_MODE3**' échouer après le premier octet transféré (un faux supplémentaire La transition du SCK a suivi chaque octet). Les mises à jour de la boîte 'DATA' de modification 'SPISLV' pour les octets transférés ont également été retardées.
- 4) Le DEL de couleur dispositif ne répertoriait pas 'B' (pour le bleu) en tant qu'option de couleur (même s'il était accepté).
- 5) Les paramètres de 'SPISLV' et 'I2CSLV' dispositifs n'étaient pas sauvegardés pour l'utilisateur '**I/O Dispositifs**' fichier.
- 6) copie '**Servo**' les instances ont échoué en raison d'un défaut '**Servo::Servo(Servo &toCopy)**' implémentation du constructeur de copie.
- 7) hors de portée '**Servo.writeMicroseconds()**' les valeurs ont été correctement détectées comme une erreur, mais les valeurs limites indiquées accompagnant le texte du message d'erreur étaient erronées.
- 8) Un débit en bauds légal de 115200 n'a pas été accepté quand chargé d'un '**I/O Dispositifs** texte fichier.
- 9) Les pin conflits électriques causées par un Curseur Analogique dispositif attaché n'ont pas toujours été détectées.
- 10) Dans de rares cas, passer un pointeur de chaîne défectueux (avec la chaîne 0-terminator manquant) à un '**String**' fonction pourrait faire planter UnoArduSim.
- 11) Le **Volet de Code** surligner pourrait-il afficher la ligne d'erreur Analyser dans le **faux** Module programme (lorsque '**#include**' a été utilisé).
- 12) Le chargement d'un dispositifs 'I/O' fichier qui avait un dispositif qui aurait (incorrectement) conduction contre 'Uno' ou 'Mega' pin 13 a provoqué un blocage de programme lors de l'affichage du message d'erreur.
- 13) UnoArduSim avait permis par erreur l'utilisateur doit coller des caractères non hexadécimaux dans le tampon d'émission agrandie fenêtres pour SPISLV et I2CSLV.
- 14) Initialisations des lignes de déclaration a échoué lorsque la valeur du côté droit était la '**return**' valeur d'un

membre objet-fonction (comme dans `int angle = myservo1.read();`).

15) '`static`' membre variables avoir explicite '`ClassName::`' les préfixes n'étaient pas reconnus s'ils apparaissaient au tout début d'une ligne (par exemple, dans une affectation à une base - 'class' variable),

16) Appel '`delete`' sur un pointeur créé par '`new`' été reconnu que si la notation fonction parenthèse était utilisée, comme dans '`delete (pptr)`'.

17) UnoArduSim implémentation de '`noTone()`' insérait à tort qu'un argument pin devait être fourni.

18) Modifications qui ont augmenté le nombre total d'octets 'RAM' dans un programme utilisé '`String`' variables (via **Modifier/Examiner** ou **Fichier | Chargez**), pourrait conduire à la corruption dans cet espace global 'Uno' ou 'Mega' en raison de la suppression du tas '`String`' objets appartenant à l'ancien programme en utilisant (incorrectement) le segment de mémoire appartenant au nouveau programme. Dans certaines circonstances, cela pourrait entraîner un crash du programme. Bien qu'un deuxième Chargez ou Analyser ait résolu le problème, ce bogue a enfin été corrigé.

19) Les valeurs de retour pour '`Wire.endTransmission()`' et '`Wire.requestFrom()`' si les deux avaient été bloqués à 0 - ceux-ci ont maintenant été corrigés.

Changements / Améliorations

V2.8.0- Juin 2020

1) Ajout du nouveau 'Mega2560' circuit imprimé **Préférences** option (Circuit imprimé numner == 10). Cette fonctionnalité beaucoup plus 'I/O' pins, 4 autres interruptions externes (pins 18, 19, 20, 21), trois autres '`HardwareSerial`' (sur pins 22-27), et la mémoire RAM est passée de 2 Ko à 8 Ko.

2) Le dispositif 'SFTSER' a été renommé 'ALTSER' (car il peut désormais également être utilisé avec 'Serial1', 'Serial2', et 'Serial3').

3) Le 'PROGIO' dispositif permet désormais une liste facultative de 4 numéros pin principaux pour définir un mappage explicite de l'esclave 4 'Uno' circuit imprimé pins au 4 maître ('Uno' ou 'Mega') circuit imprimé pins.

4) Cliquer à l'intérieur d'une boîte d'édition dispositif pin sélectionne maintenant le nombre entier pour faciliter la saisie, et cliquer à l'intérieur d'une boîte d'édition de numéro dispositifs dans Configurer | 'I/O' Dispositifs fait de même.

5) Ajout de la surbrillance orange de la ligne source pertinente pour les fenêtres contextuelles d'avertissement d'analyse et d'exécution.

6) Prise en charge supplémentaire pour '`digitalPinToInterrupt()`' appels.

7) Les classes reçoivent désormais toujours un constructeur par défaut si elles n'ont pas de constructeur spécifié par l'utilisateur (même s'ils n'ont pas de classe de base ou de membres objet).

V2.7.0 Mars 2020

1) En plus du code en ligne en cours (vert si prêt à fonctionner, rouge en cas d'erreur), UnoArduSim maintient maintenant, pour chaque module, le dernier code en ligne cliqué par l'utilisateur ou la pile de navigation (mis en évidence avec un fond d'olive foncé), la fabrication il plus facile à installer et trouver des lignes de point d'arrêt temporaires (un par module est désormais autorisé, mais seulement celui dans le module actuellement affiché est en vigueur à 'Run-To').

2) Ajout d'une nouvelle 'I/O' dispositif (et le soutien du code de la bibliothèque 3ème partie), y compris 'SPI' et 'I2C' **Port d'Expansion** 'SPI' et 'I2C' **Multiplexeur DEL** les contrôleurs et les affichages (DEL tableaux, 4-alphanumériques, et le 4-dispositifs dispositifs ou 8 afficheurs 7 segments).

- 3) '**Wire**' les opérations ne sont plus rejetées d'un utilisateur à l'intérieur d'interruption des routines (ce supporte les interruptions externes d'un 'I2C' Port d'Expansion).
- 4) Chiffre Waveforms montrent maintenant un niveau intermédiaire (entre '**HIGH**' et '**LOW**') Lorsque le analyseur est de ne pas conduit.
- 5) Pour éviter toute confusion lors par l'intensification sur simple '**SPI.transfer()**' instructions, UnoArduSim fait maintenant que vous attaché 'I/O' dispositif reçoivent maintenant leur (logique retardé) bord d'horloge 'SCK' finale avant les libre retours.
- 6) Lorsque l'auto-onglet formatage **Préférence** est activé, la saisie d'un accolade fermeture '}' dans **Modifier/Examiner** provoque maintenant un saut vers la position de retrait onglet de son ouverture accolade correspondant '{' partenaire.
- 7) UNE **Reformater** bouton a été ajouté à **Modifier/Examiner** (Pour provoquer reformatage immédiate-indent automatique onglet) - ce bouton est activé uniquement lorsque la préférence indentation automatique onglet est activée.
- 8) Un clair message d'erreur se produit maintenant, quand un mot-clé Prefix (comme '**const**', '**unsigned**', ou 'PROGMEM') suit un identificateur dans une déclaration (il doit précéder l'identificateur).
- 9) variable mondiale initialisés, même jamais utilisé plus tard, sont maintenant toujours attribuer une adresse mémoire, et ainsi sera visible.

V2.6.0 janvier 2020

- 1) affichage de caractères LCD ajouté dispositifs ayant 'SPI', 'I2C', et l'interface 4-bi-parallèle. Soutenir le code source bibliothèque a été ajoutée dans le nouveau dossier d'installation de 'include_3rdParty' (et peut être consulté à l'aide d'une directive '#include' normale) - Les utilisateurs peuvent également choisir d'écrire leur propre place fonctions à l'conduction dispositif LCD.
- 2) Volet de Code mise en évidence a été amélioré, avec des couleurs surlinger séparées pour un prêt de code en ligne, pour une erreur de code en ligne, et pour toute autre ligne de code.
- 3) Le menu Trouver et barre d'outil ('func' actions précédent-et suivant vers le bas) ne saut à la / précédent suivant la ligne de départ de fonction, et au lieu Ascend maintenant (ou descente) l'appel pile, mettant en évidence le code ligne correspondant à l'appelant (ou appelé) fonction, respectivement, où le contenu Volet de Variables est ajustée pour montrer variables pour le fonction contenant le code en ligne en surbrillance.
- 4) Pour éviter toute confusion, un Enregistrer fait à l'intérieur Modifier/Examiner provoque une réélection immédiate Compiler, et si le Enregistrer a réussi, en utilisant une Annuler ultérieure ou Quitter maintenant que du texte revert à ce dernier texte sauvegardé.
- 5) Ajout d'un Moteur Pas à Pas d'entrée pulsé ('PSTEPR') avec 'STEP' (impulsion), 'EN' (activé), et les entrées 'DIR' (sens), et un réglage micro-pas-par-étape consistant à (1,2,4,8, ou 16) .
- 6) Les deux 'STEPR' et 'PSTEPR' dispositifs ont maintenant un 'sync' DEL (VERT pour synchronisée, et rouge pour off par une ou plusieurs étapes).
- 7) 'PULSER' dispositifs ont maintenant le choix entre microsecondes et millisecondes pour 'Period' et 'Pulse'.
- 8) Intégré-fonction auto-complétions ne retiennent plus le type de paramètre devant le nom du paramètre.
- 9) Lors du retour à une Volet de Code précédente, la ligne précédemment mis en évidence est de nouveau mis en évidence aujourd'hui.
- 10) À titre d'aide à la fixation d'un point de rupture temporaire, en utilisant Annuler ou Quitter de feuilles Modifier/Examiner le surlinger dans le Volet de Code sur la dernière ligne visitées par le curseur dans Modifier/Examiner.
- 11) Un défini par l'utilisateur (ou 3ème partie) 'class' est maintenant autorisé à utiliser ou 'Print' 'Stream' que sa classe de base. À l'appui de cela, un nouveau dossier 'include_Sys' a été ajouté (dans le dossier d'installation UnoArduSim) qui fournit le code source pour chaque 'class' de base. Dans ce cas, les appels à une telle base 'class' fonctions seront traitées de manière identique au code d'utilisateur (qui) peut être entré dans), au lieu d'un

intégré fonction qui ne peut pas être entré dans (comme 'Serial.print()').

12) Membre-fonction auto-complétions comprennent maintenant le nom du paramètre **au lieu de** son type.

13) Le UnoArduSim Analyser permet désormais un nom objet dans une déclaration variables à préfacé par son 'struct' en option (et la correspondance) ou 'class' mot-clé, suivi par le 'struct' ou le nom 'class'.

V2.5.0 octobre 2019

14) Ajout du support **'TFT.h'** bibliothèque (à l'exception de **'drawBitmap()'**), Et ajouté un 'TFT' associé 'I/O' Dispositif (128 par 160 pixels). Notez que pour éviter les retards à temps réel excessives pendant les grandes **'fillXXX()'** le transfert, la partie de sa des transferts de 'SPI' **au milieu du remblai** sera absent du bus 'SPI'.

15) Pendant les grandes fichier transferts par 'SD', un partie des transferts de 'SPI' au milieu de la séquence d'octets sera absent de manière similaire à partir du bus de 'SPI'.

16) Diminué **'Stream'** les frais généraux -usage octets de sorte que **'RAM free'** valeur correspond plus étroitement Arduino compilation.

17) UnoArduSim met en garde maintenant l'utilisateur lorsqu'un **'class'** a plusieurs membres déclarés sur une ligne de déclaration.

18) L'utilisation 'File | Save As' définit maintenant le répertoire courant qui a sauvé dans le répertoire-.

19) Les deux disparus **'remove()'** membre fonctions ont été ajoutés à la **'String'** classe.

20) UnoArduSim maintenant des appels de base exclut du financement constructeur dans un constructeur-fonction échelle à moins que la définition complète du corps fonction suit immédiatement (afin de se mettre d'accord avec le Acompileur Arduino).

21) temps de transition Edge de les formes d'ondes numérique a été réduit pour soutenir la visualisation des signaux de 'SPI' rapide au plus haut zoom.

22) UnoArduSim permet maintenant certains constructeurs à déclarer **'private'** ou **'protected'** (Pour une utilisation de classe interne).

V2.4 mai 2019

1) Tous les fichiers de périphérique 'I / O' sont maintenant enregistrés dans une langue traduite. Ils sont également enregistrés avec le fichier de **Préférences**, au format UTF-8 afin d'éviter les erreurs de correspondance lors de la lecture suivante.

2) Ajout d'un nouveau **'PROGIO'** Dispositif qui est un esclave nu programmable 'Uno' ou 'Mega' circuit imprimé qui partage jusqu'à 4 pins en commun avec le **Volet du Banc de Laboratoire** maître 'Uno' ou 'Mega', - un esclave 'Uno' ou 'Mega' peut avoir non 'I/O' dispositifs à part.

3) Vous pouvez maintenant supprimer tout 'I/O' dispositif en cliquant dessus tout en appuyant sur la touche 'Ctrl'.

4) Dans **Modifier/Examiner** , l'achèvement automatique du texte a été ajouté pour les éléments globaux, intégrés et membres variables et fonctions (utilisez ALT-flèche droite pour demander un achèvement, ou **Entrer** si la liste des éléments intégrés met actuellement en surbrillance une sélection correspondante).

5) Dans **Préférences** , un nouveau choix permet l'insertion automatique d'un point-virgule de fin de ligne sur n **Entrer** key-press (si la ligne en cours est une instruction exécutable qui semble autonome et complète).

6) Pressage **'Ctrl-S'** de **Forme d'Onde** fenêtre vous permet de sauvegarder sur un fichier tous **(X, Y)** points le long de la section affichée de chaque forme d'Onde (où X est égal à la microseconde de la plus à gauche forme

d'Onde point, et Y est volts).

7) Un 'SFTSER' 'dispositif a maintenant un caché (facultatif) '**inverted**' valeur (*s'applique à la fois TX et RX*) qui peut être ajouté après sa valeur de débit en bauds à la fin de sa ligne dans un **IODevs.txt** fichier.

8) Ajouté le '**SPISettings**' classe, fonctions '**SPI.transfer16()**', '**SPI.transfer(byte* buf, int count)**', '**SPI.beginTransaction()**', et '**SPI.endTransaction()**', aussi bien que '**SPI.usingInterrupt()**' et '**SPI.notUsingInterrupt()**'.

9) Bibliothèque SPI fonctions ajoutée '**SPI.detachInterrupt()**' avec une extension de la bibliothèque SPI '**SPI.attachInterrupt(void myISRfunc)**' (au lieu de la bibliothèque actuelle fonction '**SPI.attachInterrupt(void)**' (afin d'éviter d'avoir à reconnaître des génériques '**ISR(int vector)**' déclarations d'interruptions fonction vectorielles de bas niveau).

10) Le système SPI peut maintenant être utilisé en mode esclave, soit en effectuant la '**SS**' pin (pin 10) an '**INPUT**' pin et le conduire '**LOW**' après '**SPI.begin()**' ou en précisant '**SPI_SLV**' comme optionnel '**mode**' paramètre dans '**SPI.begin(int mode=SPI_MSTR)**' (une autre extension UnoArduSim à '**SPI.h**'). Les octets reçus peuvent ensuite être collectés en utilisant '**rxbyte = SPI.transfer(tx_byte)**' soit dans un fonction d'interruption non-SPI, soit dans un service d'interruption d'utilisateur fonction précédemment associé par '**SPI.attachInterrupt(myISRfunc)**'. En mode esclave, '**transfer()**' attend qu'un octet de données soit prêt dans SPDR (bloquera donc normalement l'attente d'une réception d'octet complète, mais dans une routine d'interruption, **revenir** immédiatement car l'octet SPI reçu est déjà là). Dans tous les cas, '**tx_byte**' est placé dans le SPDR, sera donc reçu par le SPI maître attaché lors de sa prochaine '**transfer()**'.

11) La prise en charge du mode Esclave a été ajoutée à la mise en œuvre de 'Wire.h' par UnoArduSim. Fonction '**begin(uint8_t slave_address)**' est maintenant disponible, comme le sont '**onReceive(void*)**' et '**onRequest(void*)**'.

12) '**Wire.end()**' et '**Wire.setClock(freq)**' peut maintenant être appelé; ce dernier pour régler la fréquence SCL avec un '**freq**' une valeur de 100 000 (fréquence SCL par défaut en mode standard) ou de 400 000 (mode rapide).

13) 'I2CSLV' dispositifs répondent tous maintenant à la **0x00** adresse de bus d'appel général, etc. **0x00** ne peut plus être choisi comme adresse de bus I2C unique pour l'un de ces esclaves.

14) Les délais exécution modélisés des opérations de base et des opérations d'affectation et tableau et les opérations de pointeur ont été réduites et 4 microsecondes sont maintenant ajoutées pour chaque opération en virgule flottante.

V2.3 déc. 2018

1) Le suivi est maintenant activé sur le **Barre d'outils** 'I/O____S' curseur pour continu et lisse mise à l'échelle des valeurs dispositif de 'I/O' auxquelles l'utilisateur a ajouté le suffixe 'S'.

2) Un nouveau '**LED4**' 'I/O' dispositif (rangée de 4 voyants allumés) **4 numéros pin consécutifs**) a été ajouté.

3) Un nouveau '**7SEG**' 'I/O' dispositif (DEL chiffre à 7 segments avec code hexadécimal sur **4 numéros pin consécutifs** et avec active-low **CS** * sélectionner l'entrée), a été ajouté.

4) Un nouveau '**JUMP**' 'I/O' dispositif qui agit comme un cavalier entre deux 'Uno' ou 'Mega' pins a été ajouté. Cela permet une '**OUTPUT**' pin à brancher à un '**INPUT**' pin (voir le dispositif ci-dessus pour les utilisations possibles de cette nouvelle fonctionnalité).

5) Un nouveau '**OWISLV**' 'I/O' dispositif a été ajouté, et le tiers '**<OneWire.h>**' bibliothèque peut maintenant être utilisé avec '**#include**' afin que l'utilisateur programmes puisse tester l'interfaçage avec un petit sous-ensemble du bus dispositifs de '1-Wire'.

6) le **Exécuter** menu **Réinitialiser** la commande est maintenant connectée au **Réinitialiser** bouton.

7) Pour plus de clarté, quand **Retard artificiel 'loop()'** est sélectionné sous le **Options** menu, un explicite '**delay(1)**' appel est ajouté au bas de la boucle à l'intérieur '**main()**' - C'est maintenant un retard réel qui

peut être interrompu par des interruptions d'utilisateur attachées à 'Uno' ou 'Mega' pins 2 et 3.

8) Les pin électriques conflit avec à drain ouvert ou 'I/O' dispositifs sélectionnés par CS (par exemple I2CLV ou SPISLV) sont maintenant déclarés **seulement lorsqu'un vrai conflit survient à l'heure exécution**, plutôt que de provoquer une erreur immédiate lors de la première connexion du dispositif.

9) Fonction '**pulseInLong()**' est maintenant précis à 4-8 microsecondes pour être en accord avec Arduino (la précision précédente était de 250 microsecondes).

10) Erreurs signalées lors de l'initialisation d'un variable global maintenant surligner que variable dans le **Volet de Code**.

V2.2 juin 2018

1) Sur **Enregistrer** de la **Préférences** boîte de dialogue, ou de **Configurer | 'I/O' Dispositifs**, la '**myArduPrefs.txt**' Le fichier est maintenant sauvegardé dans le répertoire du programme actuellement chargé - chaque **Fichier | Chargez** puis charge automatiquement le fichier, avec son IODEvs fichier spécifié, à partir du même répertoire programme.

2) Fonction '**pulseInLong()**' **a été** manquant, mais a maintenant été ajouté (il repose sur '**micros()**' pour ses mesures).

3) Lorsqu'un utilisateur programme fait une '**#include**' d'un '***.h**' fichier, UnoArduSim essaie maintenant aussi automatiquement de charger le fichier correspondant. '***.c**' fichier **si** un correspondant '***.cpp**' fichier n'a pas été trouvé.

4) Insertion automatique d'un proche-accolade '**}**' (après chaque open-accolade '**{**') a été ajouté à **Préférences**.

5) Un nouveau **Options** choix de menu permet maintenant '**interrupts()**' être appelé de l'intérieur d'une routine d'interruption utilisateur - ceci est uniquement à des fins éducatives, dans la mesure où l'imbrication d'interruptions doit être évitée.

6) Type-casté de pointeurs vers un '**int**' la valeur est maintenant supportée (mais un message d'avertissement apparaîtra).

7) UnoArduSim prend désormais en charge les lignes programme étiquetées (par exemple, '**LabelName: count++;**' pour la commodité de l'utilisateur (mais '**goto**' est encore **interdit**)

8) Les avertissements Exécution se produisent maintenant lorsqu'un appel à '**tone()**' pourrait interférer avec le PWM actif sur pins 3 ou 11, lorsque '**analogWrite()**' interférerait avec un servo déjà actif sur le même pin, lorsqu'une arrivée de caractère en série est manquée parce que les interruptions sont actuellement désactivées, et lorsque les interruptions arriveraient si rapidement que UnoArduSim en manquera certaines.

V2.1 mars 2018

1) Affiché **Volet de Variables** Les valeurs ne sont maintenant actualisées que toutes les 30 millisecondes (et l'option Minimal peut encore réduire ce taux de rafraîchissement), mais le paramètre **VarRafraîchir** L'option de menu pour interdire la réduction de la mise à jour a été supprimée.

2) Les opérations qui ne ciblent qu'une partie des octets d'une valeur variable (telles que celles effectuées via des pointeurs) sont maintenant provoquer le changement à ce que la valeur variable soit reflétée dans la **Volet de Variables** afficher.

V2.0.1 janvier 2018

1) Arduino fonctions non documenté '**exp()**' et '**log()**' ont maintenant été ajoutés.

2) La 'SERVO' dispositifs peut maintenant être transformée en rotation continue (la largeur d'impulsion contrôle donc la vitesse au lieu de l'angle).

3) Dans **Modifier/Examiner**, un accolade de clôture '**}**' est maintenant automatiquement ajouté lorsque vous tapez une ouverture accolade '**{**' si vous avez choisi ça **Préférence**.

4) Si vous cliquez sur le **Modifier/Examiner** fenêtre barre de titre '**X**' pour quitter, vous avez maintenant la possibilité d'annuler si vous avez modifié, mais pas enregistré, le programme fichier affiché.

V2.0 septembre 2017

1) L'implémentation a été portée sur QtCreator afin que l'interface graphique présente quelques différences visuelles mineures, mais aucune différence fonctionnelle autre que certaines améliorations:

- a) La messagerie de la ligne d'état au bas du fenêtre principal et à l'intérieur du **Modifier/Examiner** boîte de dialogue, a été améliorée et un code couleur surligner a été ajouté.
- b) L'espace vertical alloué entre le **Volet de Code** et **Volet de Variables** est maintenant réglable via une barre de fractionnement glissable (mais non visible) au niveau de leur bordure partagée.
- c) Les valeurs de zone d'édition 'I/O' dispositif ne sont maintenant validées que lorsque l'utilisateur a déplacé le pointeur de la souris en dehors du dispositif - ceci évite des modifications automatiques fastidieuses pour imposer des valeurs légales pendant que l'utilisateur tape.

2) UnoArduSim prend désormais en charge plusieurs langues via **Configurer | Préférences**. L'anglais peut toujours être sélectionné, en plus de la langue des paramètres régionaux de l'utilisateur (à condition qu'une traduction fichier *.qm personnalisée pour cette langue soit présente dans le dossier UnoArduSim 'translations').

3) Le son a maintenant été modifié pour utiliser l'API audio Qt. Cela a nécessité la mise en sourdine dans certaines circonstances pour éviter les ruptures sonores et les clics gênants pendant les longs délais d'exploitation causés par les clics de souris de l'utilisateur - voir la section -Sounds pour plus de détails sur ce.

4) Pour plus de commodité, les blancs sont maintenant utilisés pour représenter une valeur 0 dans les zones d'édition dispositif-count dans **Configurer | 'I/O' Dispositifs** (vous pouvez donc maintenant utiliser la barre d'espace pour supprimer dispositifs).

5) Le non-calibré Le qualificateur (U) est maintenant facultatif sur les modèles 'PULSER', 'FUNCGEN' et '1SHOT' dispositifs (il s'agit de la valeur par défaut supposée).

6) UnoArduSim permet maintenant (en plus des valeurs numériques littérales) '**const**' variables de valeur entière, et '**enum**'